EXHIBIT 23

E.D. Tex. Case No. 2:24-CV-00064-JRG

Appendix B-10: Invalidity of U.S. Patent No. 7,784,058 in view of FreeBSD

REFERENCE: FreeBSD is an operating system that "provides the ability to partition the operating system environment, while maintaining the simplicity of the UNIX 'root' model."[1] FreeBSD has been publicly available at least as early as December 1993[2] and FreeBSD 4.0 was released and was publicly available at least as early as March 14, 2000.[3] FreeBSD anticipates and/or renders obvious, including under Plaintiff's apparent infringement theory,[4] all asserted claims[5] of U.S. Patent No. 7,784,058 (the "'058 patent") under 35 U.S.C. §§ 102 and/or 103. FreeBSD 4.0 is prior art to the '058 patent under at least 35 U.S.C. § 102(b).

The chart below provides representative examples of where each element of each claim is found within FreeBSD, including under Plaintiff's apparent construction of the asserted claims (and to the extent the claims are not found indefinite under 35 U.S.C. § 112). The cited evidence is merely illustrative, and Defendant reserves the right to cite alternative or additional evidence. To the extent Plaintiff contends that FreeBSD does not disclose any asserted claims or claim elements of the '058 Patent, it would have been obvious to combine the teachings of FreeBSD with: (1) the knowledge of one of ordinary skill in the art to show all the limitations of the claims; (2) the teachings of any of the prior art references set forth in Defendant's other invalidity charts with respect to the one or more limitations; and/or (3) the teachings of any of the prior art references set forth in the cover document of Defendant's Invalidity Contentions with respect to the '058 patent. Plaintiff has yet to identify any limitation of the asserted claims that it contends is not fully disclosed by FreeBSD, either alone or in combination with other prior art cited by Defendant and/or with the knowledge of one of ordinary skill in the art. To the extent Plaintiff makes any such contention in the future, Defendant expressly reserves the right to rebut any such contention, including by identifying additional obviousness combinations.

---

[1] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf

[2] https://web.archive.org/web/20000302095241/http://www6.freebsd.org/handbook/history.html

[3] https://web.archive.org/web/20000511211421/http://www.freebsd.org/releases/4.0R/announce.html.

[4] To the extent that these Invalidity Contentions rely on or otherwise embody particular constructions of terms or phrases in the Asserted Claims, Defendant is not proposing any such constructions as proper constructions of those terms or phrases. Various positions put forth in this document are predicated on Plaintiff's incorrect and overly broad interpretation of its claims as evidenced by its Infringement Contentions provided to Defendant. Those positions are not intended to and do not necessarily reflect Defendant's interpretation of the true and proper scope of Plaintiff's claims, and Defendant reserves the right to adopt claim construction positions that differ from or even conflict with various positions put forth in this document.

[5] As used herein, "asserted claims" refers only to those claims charted in Plaintiff's July 1, 2024 Infringement Contentions. To the extent Plaintiff later obtains leave to assert any additional claims for this patent, Defendant will provide its preliminary invalidity contentions consistent with the timing requirements set forth in the Court's order.

Chart B-10

Invalidity Contentions: FreeBSD

Where the chart below states that FreeBSD "discloses" a limitation, such disclosure may be express or inherent. All emphasis is added unless otherwise indicated.

Fact Discovery is ongoing and Defendant's prior art investigation, including via third party discovery, is therefore not yet complete. Defendant reserves the right to rely upon additional evidence of invalidity obtained in the future as to FreeBSD or any other prior art public use/sales/offers for sale that may anticipate or render obvious one or more asserted claims of the Asserted Patents under 35 U.S.C. § 102(b) and/or 35 U.S.C. § 103. Defendant may rely on addition documents and/or things that have not yet been located and/or testimony to support the contentions regarding FreeBSD set forth in this chart.

Claim 1

'058 Patent Claim 1	**Disclosure**
1[p] A computing system for executing a plurality of software applications comprising:	FreeBSD, as evidenced by the example citations below, discloses a computing system for executing a plurality of software applications comprising, as this claim limitation appears to be interpreted by VirtaMove. *See, e.g.:* • "FreeBSD is a 4.4BSD-Lite2 based operating system for the Intel architecture (x86) and DEC Alpha based systems."[6] • "FreeBSD is a state-of-the-art operating system based on 4.4BSD-Lite2. It runs on computer systems based on the Intel architecture (x86), and also the DEC Alpha architecture."[7] • "FreeBSD has many noteworthy features. Some of these are: . . . *Preemptive multitasking* with dynamic priority adjustment to ensure smooth and fair sharing of the computer between applications and users, even under the heaviest of loads. . . . *Multiuser facilities* which allow many people to use a FreeBSD system simultaneously for a variety of things. This means, for example, that system peripherals such as printers and tape drives are properly shared between all users on the system or the network and that individual resource limits can be placed on users or groups of users, protecting critical system resources from over-use. . . . *Memory protection* ensures that applications (or users) cannot interfere with each other. One application crashing will not affect others in any way. . . .Thousands of *ready-to-run* applications are available from the FreeBSD ports and packages collection. . . . Thousands of additional and *easy-to-port* applications available on the Internet. FreeBSD is source code compatible with most popular commercial Unix systems and thus most applications require few, if any, changes to compile. . . . Demand paged *virtual memory* and 'merged VM/buffer cache' design efficiently satisfies applications with large appetites for memory while still maintaining interactive response to other users."[8] • "FreeBSD is a 32-bit operating system (64-bit on the Alpha) and was designed as such from the ground up."[9] • "The applications to which FreeBSD can be put are truly limited only by your own imagination. From software development to factory automation, inventory control to azimuth correction of remote satellite antennae; if it can be done with a commercial UNIX product then it is more than likely that you can do it with FreeBSD, too! FreeBSD also benefits significantly from the literally thousands of high quality

[6] https://web.archive.org/web/20000815235527/http://www.freebsd.org:80/handbook/introduction.html
[7] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html
[8] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html (emphasis in original)
[9] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html

Chart B-10 **Invalidity Contentions: FreeBSD**

’058 Patent Claim 1	Disclosure
	applications developed by research centers and universities around the world, often available at little to no cost. Commercial applications are also available and appearing in greater numbers every day.”[10] • “Because the source code for FreeBSD itself is generally available, the system can also be customized to an almost unheard of degree for special applications or projects, and in ways not generally possible with operating systems from most major commercial vendors.”[11] • “Unlike an X terminal, FreeBSD allows many applications to be run locally, if desired, thus relieving the burden on a central server. FreeBSD can even boot ‘diskless’, making individual workstations even cheaper and easier to administer.”[12] • “FreeBSD includes many applications and utilities produced by the Free Software Foundation (FSF).”[13] • “The FreeBSD Ports collection allows you to compile and install a very wide range of applications with a minimum amount of effort.”[14] • “FreeBSD is a liberally-licensed open source operating system with its origins in BSD Net/2 and 4.4 Lite, the Berkeley Software Distributions developed at the University of California at Berkeley until 1994.”[15] • .“FreeBSD can run a wide variety of software packages, most of which are available as source code so they can be built as native FreeBSD software. And, thanks to some clever design, FreeBSD can also run software from many foreign operating systems. We'll look at how to do this, focusing on the popular Linux compatibility package that allows FreeBSD to run unmodified Linux software.”[16] To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’058 Patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.
1[a] a) a processor,	FreeBSD, as evidenced by the example citations below, discloses a processor, as this claim limitation appears to be interpreted by VirtaMove. *See, e.g.:*

[10] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html
[11] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html
[12] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html
[13] https://web.archive.org/web/20000815235602/http://www.freebsd.org/handbook/x1353.html
[14] https://web.archive.org/web/20000815234714/http://www.freebsd.org/handbook/ports.html
[15] https://lists.freebsd.org/pipermail/freebsd-announce/2000-October/000595.html
[16] Lucas, M., Absolute BSD: the Ultimate Guide to Free BSD 250 (2002)

Chart B-10 **Invalidity Contentions: FreeBSD**

’058 Patent Claim 1	Disclosure
	• “FreeBSD has many noteworthy features. Some of these are: . . . SMP support for machines with multiple CPUs (Intel only).”[17] • “With FreeBSD, you can easily start out small with an inexpensive 386 class PC and upgrade all the way up to a quad-processor Xeon with RAID storage as your enterprise grows.”[18] • “FreeBSD currently runs on a wide variety of ISA, VLB, EISA, and PCI bus based PCs, ranging from the 386SX to Pentium class machines (though the 386SX is not recommended). Support for generic IDE or ESDI drive configurations, various SCSI controllers, and network and serial cards is also provided.”[19] • “2.3. Supported Hardware . . . 2.3.1. Disk Controllers ○ WD1003 (any generic MFM/RLL) ○ WD1007 (any generic IDE/ESDI) ○ IDE ○ ATA ○ Adaptec 1535 ISA SCSI controllers ○ Adaptec 154X series ISA SCSI controllers ○ Adaptec 174X series EISA SCSI controllers in standard and enhanced mode ○ Adaptec 274X/284X/2920C/294X/2950/3940/3950 (Narrow/Wide/Twin) series EISA/VLB/PCI SCSI controllers ○ Adaptec AIC-7850, AIC-7860, AIC-7880, AIC-789X on-board SCSI controllers ○ Adaptec 1510 series ISA SCSI controllers (not for bootable devices) ○ Adaptec 152X series ISA SCSI controllers ○ Adaptec AIC-6260 and AIC-6360 based boards, which include the AHA-152X and SoundBlaster SCSI cards ○ AdvanSys SCSI controllers (all models) ○ BusLogic MultiMaster ‘W’ Series Host Adapters including BT-948, BT-958, BT-9580 ○ BusLogic MultiMaster ‘C’ Series Host Adapters including BT-946C, BT-956C, BT-956CD, BT-445C, BT-747C, BT-757C, BT-757CD, BT-545C, BT-540CF ○ BusLogic MultiMaster ‘S’ Series Host Adapters including BT-445S, BT-747S, BT-747D, BT-757S, BT-757D, BT-545S, BT-542D, BT-742A, BT-542B ○ BusLogic MultiMaster ‘A’ Series Host Adapters including BT-742A, BT-542B ○ AMI FastDisk controllers that are true BusLogic MultiMaster clones are also supported. ▪ Note: BusLogic/Mylex ‘Flashpoint’ adapters are NOT yet supported. ○ DPT SmartCACHE Plus, SmartCACHE III, SmartRAID III, SmartCACHE IV, and SmartRAID IV SCSI/RAID are supported. The DPT SmartRAID/CACHE V is not yet supported.

[17] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html
[18] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html
[19] https://web.archive.org/web/20000510155715/http://www.freebsd.org:80/handbook/install-hw.html

’058 Patent Claim 1	Disclosure
	○ Compaq Intelligent Disk Array Controllers: IDA, IDA-2, IAES, SMART, SMART-2/E, Smart-2/P, SMART-2SL, Integrated Array, and Smart Arrays 3200, 3100ES, 221, 4200, 4200, 4250ES. ○ SymBios (formerly NCR) 53C810, 53C810a, 53C815, 53C820, 53C825a, 53C860, 53C875, 53C875j, 53C885, and 53C896 PCI SCSI controllers including ASUS SC-200, Data Technology DTC3130 (all variants), Diamond FirePort (all), NCR cards (all), SymBios cards (all), Tekram DC390W, 390U, and 390F, and Tyan S1365 ○ QLogic 1020, 1040, 1040B, and 2100 SCSI and Fibre Channel Adapters ○ DTC 3290 EISA SCSI controller in 1542 evaluation mode”[20] • “With all supported SCSI controllers, full support is provided for SCSI-I and SCSI-II peripherals, including hard disks, optical disks, tape drives (including DAT and 8mm Exabyte), medium changers, processor target devices, and CDROM drives. WORM devices that support CDROM commands are supported for read-only access by the CDROM driver. WORM/CD-R/CD-RW writing support is provided by cdrecord, which is in the ports tree.”[21] • “2.3. Supported Hardware . . . 2.3.2. Network Card ○ Adaptec Duralink PCI fast ethernet adapters based on the Adaptec AIC-6195 fast ethernet controller chip, including the following: ▪ ANA-62011 64-bit single port 10/100baseTX adapter ▪ ANA-62022 64-bit dual port 10/100baseTX adapter ▪ ANA-62044 64-bit quad port 10/100baseTX adapter ▪ ANA-69011 32-bit single port 10/100baseTX adapter ▪ ANA-62020 64-bit single port 100baseFX adapter ○ Allied-Telesyn AT1700 and RE2000 cards ○ Alteon Networks PCI gigabit ethernet NICs based on the Tigon 1 and Tigon 2 chipsets including the Alteon AceNIC (Tigon 1 and 2), 3Com 3c985-SX (Tigon 1 and 2), Netgear GA620 (Tigon 2), Silicon Graphics Gigabit Ethernet, DEC/Compaq EtherWORKS 1000, NEC Gigabit Ethernet ○ AMD PCnet/PCI (79c970 and 53c974 or 79c974) ○ RealTek 8129/8139 fast ethernet NICs including the following: ▪ Allied-Telesyn AT2550 ▪ Allied-Telesyn AT2500TX ▪ Genius GF100TXR (RTL8139) ▪ NDC Communications NE100TX-E ▪ OvisLink LEF-8129TX ▪ OvisLink LEF-8139TX ▪ Netronix Inc. EA-1210 NetEther 10/100 ▪ KTX-9130TX 10/100 Fast Ethernet

[20] https://web.archive.org/web/20000510155715/http://www.freebsd.org:80/handbook/install-hw.html
[21] https://web.archive.org/web/20000510155715/http://www.freebsd.org:80/handbook/install-hw.html

Chart B-10 **Invalidity Contentions: FreeBSD**

’058 Patent Claim 1	Disclosure
	▪ Accton ‘Cheetah’ EN1027D (MPX 5030/5038; RealTek 8139 clone?) ▪ SMC EZ Card 10/100 PCI 1211-TX o Lite-On 98713, 98713A, 98715, and 98725 fast ethernet NICs, including the LinkSys EtherFast LNE100TX, NetGear FA310-TX Rev. D1, Matrox FastNIC 10/100, Kingston KNE110TX o Macronix 98713, 98713A, 98715, 98715A, and 98725 fast ethernet NICs including the NDC Communications SFA100A (98713A), CNet Pro120A (98713 or 98713A), CNet Pro120B (98715), SVEC PN102TX (98713) o Macronix/Lite-On PNIC II LC82C115 fast ethernet NICs including the LinkSys EtherFast LNE100TX version 2 o Winbond W89C840F fast ethernet nics including the Trendware TE100-PCIE o VIA Technologies VT3043 ‘Rhine I’ and VT86C100A ‘Rhine II’ fast ethernet NICs including the Hawking Technologies PN102TX and D-Link DFE-530TX o Silicon Integrated Systems SiS 900 and SiS 7016 PCI fast ethernet NICs o Sundance Technologies ST201 PCI fast ethernet NICs including the D-Link DFE-550TX o SysKonnect SK-984x PCI gigabit ethernet cards including the SK-9841 1000baseLX (single mode fiber, single port), the SK-9842 1000baseSX (multimode fiber, single port), the SK-9843 1000baseLX (single mode fiber, dual port), and the SK-9844 1000baseSX (multimode fiber, dual port). o Texas Instruments ThunderLAN PCI NICs, including the Compaq Netelligent 10, 10/100, 10/100 Proliant, 10/100 Dual-Port, 10/100 TX Embedded UTP, 10 T PCI UTP/Coax, and 10/100 TX UTP, the Compaq NetFlex 3P, 3P Integrated, and 3P w/BNC, the Olicom OC-2135/2138, OC-2325, OC-2326 10/100 TX UTP, and the Racore 8165 10/100baseTX and 8148 10baseT/100baseTX/100baseFX multi-personality cards o ADMtek AL981-based and AN985-based PCI fast ethernet NICs o ASIX Electronics AX88140A PCI NICs including the Alfa Inc. GFC2204 and CNet Pro110B o DEC EtherWORKS III NICs (DE203, DE204, and DE205) o DEC EtherWORKS II NICs (DE200, DE201, DE202, and DE422) o DEC DC21040, DC21041, or DC21140 based NICs (SMC Etherpower 8432T, DE245, etc.) o DEC FDDI (DEFPA/DEFEA) NICs o Efficient ENI-155p ATM PCI o FORE PCA-200E ATM PCI o Fujitsu MB86960A/MB86965A o HP PC Lan+ cards (model numbers: 27247B and 27252A) o Intel EtherExpress (not recommended due to driver instability) o Intel EtherExpress Pro/10 o Intel EtherExpress Pro/100B PCI Fast Ethernet o Isolan AT 4141-0 (16 bit) o Isolink 4110 (8 bit)

’058 Patent Claim 1	Disclosure
	○ Novell NE1000, NE2000, and NE2100 Ethernet interfaces ○ PCI network cards emulating the NE2000, including the RealTek 8029, NetVin 5000, Winbond W89C940, Surecom NE-34, VIA VT86C926 ○ 3Com 3C501, 3C503 Etherlink II, 3C505 Etherlink/+, 3C507 Etherlink 16/TP, 3C509, 3C579, 3C589 (PCMCIA), 3C590/592/595/900/905/905B/905C PCI and EISA (Fast) Etherlink III / (Fast) Etherlink XL, 3C980/3C980B Fast Etherlink XL server adapter, 3CSOHO100-TX OfficeConnect adapter ○ Toshiba ethernet cards ○ PCMCIA ethernet cards from IBM and National Semiconductor are also supported”[22] • “2.3. Supported Hardware . . . 2.3.3. USB Peripherals . . . ○ Motherboard chipsets: ▪ ALi Aladdin-V ▪ Intel 82371SB (PIIX3) and 82371AB and EB (PIIX4) chipsets ▪ NEC uPD 9210 Host Controller ▪ VIA 83C572 USB Host Controller ▪ and any other UHCI or OHCI compliant motherboard chipset (no exceptions known).”[23] • “The system is responsible for properly sharing and managing requests for hardware devices, peripherals, memory, and CPU time evenly to each user.”[24] • “FreeBSD uses a three-stage bootstrap by default, which basically entails three programs which call each other in order (two boot blocks, and the loader). Each of these three build on the previous program's understanding and provide increasing amounts of sophistication. The kernel is then started, which will then probe for devices and initialize them for use. Once the kernel boot process is finished, the kernel passes control to the user process init(8), which then makes sure the disks are in a usable state. init(8) then starts the user-level resource configuration which then mounts filesystems, sets up network cards to act on the network, and generally starts all the processes that usually are run on a FreeBSD system at startup.”[25] • “This is the machine architecture. It must be either i386, alpha, or pc98. cpu I386_CPU cpu I486_CPU cpu I586_CPU cpu I686_CPU

[22] https://web.archive.org/web/20000510155715/http://www.freebsd.org:80/handbook/install-hw.html
[23] https://web.archive.org/web/20000510155715/http://www.freebsd.org:80/handbook/install-hw.html
[24] https://web.archive.org/web/20000622040046/http://www.freebsd.org/handbook/permissions.html
[25] https://web.archive.org/web/20000815235626/http://www.freebsd.org/handbook/boot.html

’058 Patent Claim 1	Disclosure
	The above specifies the type of CPU you have in your system. You may have multiple instances of the CPU line (i.e., you are not sure whether you should use I586_CPU or I686_CPU), however, for a custom kernel, it is best to specify only the CPU you have. If you are unsure which type your CPU use, you can use the dmesg command to view your boot up messages. The Alpha architecture has different values for cpu_type. They include: cpu EV4 cpu EV5 If you are using an Alpha machine, you should be using one of the above CPU types. ident GENERIC.”[26] • “FreeBSD also supports the Alpha processor, and work is underway to support Intel's new IA64, AMD's new 64−bit chips, and Motorola's PowerPC, as well as Sun's SPARC.”[27] • “Your brand of processor is really irrelevant to FreeBSD; FreeBSD won't care if you're running an Intel, AMD, IBM, or Cyrix CPU. It probes the CPU on booting, and uses whatever chip features it recognizes. I've run effective servers on 486 machines before—in fact, I've filled a T1 Internet circuit with a 486. However, I would still recommend that you get a Pentium or faster CPU. Some of the demonstrations in this book take less than an hour on my twin 1 GHz Pentium system, but take almost three days on my ancient 25 MHz 486.” [28] To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’058 Patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.
1[b] b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,	FreeBSD, as evidenced by the example citations below, discloses an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor, as this claim

[26] https://web.archive.org/web/20000815235740/http://www.freebsd.org/handbook/kernelconfig-config.html
[27] Lucas, M., Absolute BSD: the Ultimate Guide to Free BSD 10 (200)
[28] Lucas, M., Absolute BSD: the Ultimate Guide to Free BSD 10 (200)

’058 Patent Claim 1	Disclosure
	limitation appears to be interpreted by VirtaMove. Operating systems with OS kernels with OS critical system elements for running in kernel mode were well known in the art at the time of the invention.[29] *See, e.g.:* • “FreeBSD is a 4.4BSD-Lite2 based operating system for the Intel architecture (x86) and DEC Alpha based systems.”[30] • “FreeBSD is a state-of-the-art operating system based on 4.4BSD-Lite2. It runs on computer systems based on the Intel architecture (x86), and also the DEC Alpha architecture.”[31] • “FreeBSD is a 32-bit operating system (64-bit on the Alpha) and was designed as such from the ground up.”[32] • “Because the source code for FreeBSD itself is generally available, the system can also be customized to an almost unheard of degree for special applications or projects, and in ways not generally possible with operating systems from most major commercial vendors.”[33] • “FreeBSD has many noteworthy features. Some of these are: . . . *Preemptive multitasking* with dynamic priority adjustment to ensure smooth and fair sharing of the computer between applications and users, even under the heaviest of loads. . . . *Multiuser facilities* which allow many people to use a FreeBSD system simultaneously for a variety of things. This means, for example, that system peripherals such as printers and tape drives are properly shared between all users on the system or the network and that individual resource limits can be placed on users or groups of users, protecting critical system resources from over-use.”[34] • “In order to run FreeBSD, a recommended minimum of eight megabytes of RAM is suggested. Sixteen megabytes is the preferred amount of RAM as you may have some trouble with anything less than sixteen depending on your hardware.”[35] • “FreeBSD, having its history rooted in BSD UNIX, has its fundamentals based on several key UNIX concepts. The first, and most pronounced, is that FreeBSD is a multi-user operating system. The system can handle several users all working simultaneously on completely unrelated tasks. The system is responsible for properly sharing and managing requests for hardware devices, peripherals, memory, and CPU time evenly to each user.”[36]

[29] https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/_kernel/
[30] https://web.archive.org/web/20000815235527/http://www.freebsd.org:80/handbook/introduction.html
[31] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html
[32] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html
[33] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html
[34] https://web.archive.org/web/20000815235532/http://www.freebsd.org/handbook/nutshell.html
[35] https://web.archive.org/web/20000815233005/http://www.freebsd.org/handbook/install-hw.html
[36] https://web.archive.org/web/20000622040046/http://www.freebsd.org/handbook/permissions.html

’058 Patent Claim 1	Disclosure
	• “FreeBSD uses a three-stage bootstrap by default, which basically entails three programs which call each other in order (two boot blocks, and the loader). Each of these three build on the previous program's understanding and provide increasing amounts of sophistication. The kernel is then started, which will then probe for devices and initialize them for use. Once the kernel boot process is finished, the kernel passes control to the user process init(8), which then makes sure the disks are in a usable state. init(8) then starts the user-level resource configuration which then mounts filesystems, sets up network cards to act on the network, and generally starts all the processes that usually are run on a FreeBSD system at startup.”[37] • “The easy-to-use command set comprises of: autoboot seconds Proceeds to boot the kernel if not interrupted within the time span given, in seconds. It displays a countdown, and the default timespan is 10 seconds. boot [-options] [kernelname] Immediately proceeds to boot the kernel, with the given options, if any, and with the kernel name given, if it is. boot-conf Goes through the same automatic configuration of modules based on variables as what happens at boot. This only makes sense if you use unload first, and change some variables, most commonly kernel. help [topic] Shows help messages read from /boot/loader.help. If the topic given is index, then the list of available topics is given. include filename ... Processes the file with the given filename. The file is read in, and interpreted line by line. An error immediately stops the include command. load [-t type] filename Loads the kernel, kernel module, or file of the type given, with the filename given. Any arguments after filename are passed to the file. ls [-l] [path] Displays a listing of files in the given path, or the root directory, if the path is not specified. If -l is specified, file sizes will be shown too. lsdev [-v] Lists all of the devices from which it may be possible to load modules. If -v is specified, more details are printed. lsmod [-v] Displays loaded modules. If -v is specified, more details are shown.

[37] https://web.archive.org/web/20000815235626/http://www.freebsd.org/handbook/boot.html

’058 Patent Claim 1	Disclosure
	more filename Display the files specified, with a pause at each LINES displayed. reboot Immediately reboots the system. set variable, set variable=value Set loader's environment variables. unload Removes all loaded modules.”[38] • “Here are some practical examples of loader usage. ○ To simply boot your usual kernel, but in single-user mode: boot -s ○ To unload your usual kernel and modules, and then load just your old (or another) kernel: unload load kernel.old You can use kernel.GENERIC to refer to the generic kernel that comes on the install disk, or kernel.old to refer to your previously installed kernel (when you've upgraded or configured your own kernel, for example). Note: Use the following to load your usual modules with another kernel: unload set kernel=’kernel.old’ boot-conf ○ To load a kernel configuration script (an automated script which does the things you'd normally do in the kernel boot-time configurator): load -t userconfig_script /boot/kernel.conf”[39] • “5.4. Kernel Interaction During Boot . . . Once the kernel is loaded by either loader (as usual) or boot2 (bypassing the loader), it examines its boot flags, if any, and adjusts its behavior as necessary.”[40]

[38] https://web.archive.org/web/20000815235638/http://www.freebsd.org/handbook/boot-loader.html
[39] https://web.archive.org/web/20000815235638/http://www.freebsd.org/handbook/boot-loader.html
[40] https://web.archive.org/web/20000815235643/http://www.freebsd.org/handbook/boot-kernel.html

’058 Patent Claim 1	Disclosure
	• “5.4.1. Kernel Boot Flags Here are the more common boot flags: -a during kernel initialization, ask for the device to mount as [] the root file system. -C boot from CDROM. -c run UserConfig, the boot-time kernel configurator -s boot into single-user mode -v be more verbose during kernel startup”[41] • “Building a custom kernel is one of the most important rites of passage nearly every UNIX user must endure. This process, while time consuming, will provide many benefits to your FreeBSD system. Unlike the GENERIC kernel, which must support a wide range of hardware, a custom kernel only contains support for your PC’s hardware. . . . Faster boot time. Since the kernel will only probe the hardware you have on your system, the time it takes your system to boot will decrease dramatically. . . . Less memory use. A custom kernel often uses less memory than the GENERIC kernel, which is important because the kernel is one process that must always be present in memory. For this reason, a custom kernel is especially useful on a system with a small amount of RAM. [42] • “First, let us take a quick tour of the kernel build directory. All directories mentioned will be relative to the main /usr/src/sys directory, which is also accessible through /sys. There are a number of subdirectories here representing different parts of the kernel, but the most important, for our purposes, are arch/conf, where you will edit your custom kernel configuration, and compile, which is the staging area where your kernel will be built. arch represents either i386, alpha, or pc98 (an alternative development branch of PC hardware, popular in Japan). Everything inside a particular architecture's directory deals with that architecture only; the rest of the code is common to all platforms to which FreeBSD could potentially be ported. Notice the logical organization of the directory structure, with each supported device, filesystem, and option in its own subdirectory.”[43] • “The following are the mandatory keywords required in every kernel you build:

[41] https://web.archive.org/web/20000815235643/http://www.freebsd.org/handbook/boot-kernel.html
[42] https://web.archive.org/web/20000815235727/http://www.freebsd.org/handbook/x2428.html
[43] https://web.archive.org/web/20000815235734/http://www.freebsd.org/handbook/kernelconfig-building.html

’058 Patent Claim 1	Disclosure
	machine i386 This is the machine architecture. It must be either i386, alpha, or pc98. cpu I386_CPU cpu I486_CPU cpu I586_CPU cpu I686_CPU The above specifies the type of CPU you have in your system. You may have multiple instances of the CPU line (i.e., you are not sure whether you should use I586_CPU or I686_CPU), however, for a custom kernel, it is best to specify only the CPU you have. If you are unsure which type your CPU use, you can use the dmesg command to view your boot up messages. The Alpha architecture has different values for cpu_type. They include: cpu EV4 cpu EV5 If you are using an Alpha machine, you should be using one of the above CPU types. ident GENERIC . . .”[44] • “Your system should have fast access to its root filesystem (/), which contains the kernel and just enough utilities and programs to boot the computer into its most basic running status, single−user mode.”[45] • “The FreeBSD kernel can be dynamically tuned, or changed on the fly, and most aspects of system performance can be changed as needed. We'll discuss the kernel's sysctl interface, and how you can use it to alter a running kernel.” [46] • “The actual kernel is a file on disk: /kernel. Kernel modules—the kernel code that can be loaded and unloaded after boot—lives in /modules. Kernel modules are required in this day of detachable hardware, such as PC Cards and USB, and they can also provide additional functionality that you don't want to permanently add to the kernel.Every file you see outside of /kernel and /modules is not part of the kernel;

[44] https://web.archive.org/web/20000815235740/http://www.freebsd.org/handbook/kernelconfig-config.html
[45] Lucas, M., Absolute BSD: the Ultimate Guide to Free BSD 25 (2002)
[46] Lucas, M., Absolute BSD: the Ultimate Guide to Free BSD 70 (2002)

’058 Patent Claim 1	Disclosure
	these files and programs as a group are called the user–land, meaning they're meant for users. But at the same time, these programs and data use the kernel facilities.”[47] • “Some kernel modules don't need to be loaded in your system at boottime; they can be loaded and unloaded while the system is running. We'll look at how you can find out what modules you have in your system, then how to load and unload them. . . . If all possible functions were compiled into the kernel, the kernel would be much larger than it is. This way, you can have a smaller, more efficient kernel and only load modules as you need them.”[48] • “When you customize your kernel, you'll get better performance, and you can also include new functionality in it, or support for new hardware.”[49] • “Editing Kernel Files . . . You've now backed up your working kernel and are ready to build a new one. To begin, check out /sys/i386/conf, where you should find several files. The important ones for your purposes are GENERIC and LINT. GENERIC is the kernel configuration file for the standard kernel used on first install. LINT contains all kernel options and the documentation for them, including a variety of really obscure ones.”[50] • “Because the GENERIC kernel is designed to run on the greatest variety of equipment, it includes a huge array of network drivers, disk drivers, controllers, and features.”[51] • “Adding to the Kernel . . . At this point, if everything has gone well, you should have a minimal kernel that works well. Now you can add features and tweak it. LINT . . . You'll find a list of all kernel features in the file /sys/i386/conf/LINT, including every kernel option and driver, as well as some documentation. If you have hardware that doesn't appear to be supported in the GENERIC kernel, take a look at LINT. Some of these features are obscure, but if you have the hardware, you'll appreciate them. For example, FreeBSD supports the special features of the IBM BlueLightning CPU, which will allow both of you BlueLightning owners to use your CPU to its full extent.”[52] • “Network mbuf clusters are preallocated in kernel memory, so you can't just crank this value up to a million and forget about it, because that memory won't be available for other uses when the system gets busy. You do want your kernel to be able to open files and support your Web server, don't you?”[53] To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the

[47] Lucas, M., Absolute BSD: the Ultimate Guide to Free BSD 70 (2002)
[48] Lucas, M., Absolute BSD: the Ultimate Guide to Free BSD 78-79 (2002)
[49] Lucas, M., Absolute BSD: the Ultimate Guide to Free BSD 79 (2002)
[50] Lucas, M., Absolute BSD: the Ultimate Guide to Free BSD 80 (2002)
[51] Lucas, M., Absolute BSD: the Ultimate Guide to Free BSD 81 (2002)
[52] Lucas, M., Absolute BSD: the Ultimate Guide to Free BSD 92 (2002)
[53] Lucas, M., Absolute BSD: the Ultimate Guide to Free BSD 94 (2002)

’058 Patent Claim 1	Disclosure
	teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’058 Patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.
1[c] c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and	FreeBSD, as evidenced by the example citations below, discloses a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of Software applications in user mode, as this claim limitation appears to be interpreted by VirtaMove. *See, e.g.:* • “FreeBSD has many noteworthy features. Some of these are: . . . *Preemptive multitasking* with dynamic priority adjustment to ensure smooth and fair sharing of the computer between applications and users, even under the heaviest of loads. . . . *Multiuser facilities* which allow many people to use a FreeBSD system simultaneously for a variety of things. This means, for example, that system peripherals such as printers and tape drives are properly shared between all users on the system or the network and that individual resource limits can be placed on users or groups of users, protecting critical system resources from over-use. . . . *Memory protection* ensures that applications (or users) cannot interfere with each other. One application crashing will not affect others in any way. . . .Thousands of *ready-to-run* applications are available from the FreeBSD ports and packages collection. . . . Thousands of additional and *easy-to-port* applications available on the Internet. FreeBSD is source code compatible with most popular commercial Unix systems and thus most applications require few, if any, changes to compile. . . . Demand paged *virtual memory* and ‘merged VM/buffer cache’ design efficiently satisfies applications with large appetites for memory while still maintaining interactive response to other users.”[54] • “Since our release of FreeBSD 2.0 in late 94, the performance, feature set, and stability of FreeBSD has improved dramatically. The largest change is a revamped virtual memory system with a merged VM/file buffer cache that not only increases performance, but also reduces FreeBSD's memory footprint, making a 5MB configuration a more acceptable minimum.”[55] • “In addition to the base distributions, FreeBSD offers a ported software collection with thousands of commonly sought-after programs. By mid-January 2000, there were nearly 3000 ports! The list of ports ranges from http (WWW) servers, to games, languages, editors, and almost everything in between. The entire ports collection requires approximately 50MB of storage, all ports being expressed as ‘deltas’ to their original sources. This makes it much easier for us to update ports, and greatly reduces the disk space demands made by the older 1.0 ports collection. To compile a port, you simply change to the directory of

[54] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html (emphasis in original)
[55] https://web.archive.org/web/20000815230603/http://www.freebsd.org/handbook/history.html

’058 Patent Claim 1	Disclosure
	the program you wish to install, type make install, and let the system do the rest. The full original distribution for each port you build is retrieved dynamically off the CDROM or a local FTP site, so you need only enough disk space to build the ports you want. Almost every port is also provided as a pre-compiled ‘package’, which can be installed with a simple command (pkg_add) by those who do not wish to compile their own ports from source.”[56] • “FreeBSD, having its history rooted in BSD UNIX, has its fundamentals based on several key UNIX concepts. The first, and most pronounced, is that FreeBSD is a multi-user operating system. The system can handle several users all working simultaneously on completely unrelated tasks. The system is responsible for properly sharing and managing requests for hardware devices, peripherals, memory, and CPU time evenly to each user.”[57] • “Because the system is capable of supporting multiple users, everything the system manages has a set of permissions governing who can read, write, and execute the resource. These permissions are stored as an octet broken into three pieces, one for the owner of the file, one for the group that the file belongs to, and one for everyone else. . . . This is all well and good, but how does the system control permissions on devices? FreeBSD actually treats most hardware devices as a file that programs can open, read, and write data to just like any other file. These special device files are stored on the /dev directory. Directories are also treated as files. They have read, write, and execute permissions. The executable bit for a directory has a slightly different meaning than that of files. When a directory is marked executable, it means it can be searched into, for example, a directory listing can be done in that directory.”[58] • “Since FreeBSD uses its file systems to determine many fundamental system operations, the hierarchy of the file system is extremely important. Due to the fact that the hier(7) man page provides a complete description of the directory structure, it will not be duplicated here. Please read hier(7) for more information. Of significant importance is the root of all directories, the / directory. This directory is the first directory mounted at boot time and it contains the base system necessary at boot time. The root directory also contains mount points for every other file system that you want to mount. A mount point is a directory where additional file systems can be grafted onto the root file system. Standard mount points include /usr, /var, /mnt, and /cdrom. These directories are usually referenced to entries in the file /etc/fstab. /etc/fstab is a table of various file systems and mount points for reference by the system. Most of the file systems in /etc/fstab are mounted automatically at boot time from the script rc(8) unless they contain the noauto option. Consult the fstab(5) manual page for more information on the format of the /etc/fstab file and the options it contains.”[59]

[56] https://web.archive.org/web/20000815230603/http://www.freebsd.org/handbook/history.html
[57] https://web.archive.org/web/20000622040046/http://www.freebsd.org/handbook/permissions.html
[58] https://web.archive.org/web/20000622040046/http://www.freebsd.org/handbook/permissions.html
[59] https://web.archive.org/web/20000621185132/http://www.freebsd.org/handbook/dirstructure.html

Chart B-10 **Invalidity Contentions: FreeBSD**

’058 Patent Claim 1	Disclosure

	• “The FreeBSD Ports collection allows you to compile and install a very wide range of applications with a minimum amount of effort. In general, it is a group of skeletons which contain a minimal set of items needed to make an application compile and install cleanly on FreeBSD. Even with all the hype about open standards, getting a program to compile on various UNIX platforms can be a tricky task. Occasionally, you might be lucky enough to find that the program you want compiles cleanly on your system, install everything into all the right directories, and run flawlessly ‘out-of-the-box’, but this behavior is somewhat rare. Most of the time, you find yourself needing to make modifications in order to get the program to work. This is where the FreeBSD Ports collection comes to the rescue. The general idea behind the Ports collection is to eliminate all of the messy steps involved with making things work properly so that the installation is simple and very painless. With the Ports collection, all of the hard work has already been done for you, and you are able to install any of the Ports collection ports by simply typing make install.”[60]
	• “The first thing that should be explained when it comes to the Ports collection is what is actually meant by a ‘skeleton’. In a nutshell, a port skeleton is a minimal set of files that are needed for a program to compile and install cleanly on FreeBSD. Each port skeleton includes:
	  ○ A Makefile. The Makefile contains various statements that specify how the application should be compiled and where it should be installed on your system
	  ○ A files directory. The files directory contains a file named md5. This file is named after the MD5 algorithm used to determine ports checksums. A checksum is a number generated by adding up all the data in the file you want to check. If any characters change, the checksum will differ from the original and an error message will be displayed so you are able to investigate the changes.
	  ○ The files directory can also contain other files that are required by the port but do not belong elsewhere in the directory structure.
	  ○ A patches directory. This directory contains patches to make the program compile and install on your FreeBSD system. Patches are basically small files that specify changes to particular files. They are in plain text format, and basically say ‘Remove line 10’ or ‘Change line 26 to this ...’. Patches are also known as ‘diffs’ because they are generated by the diff program.
	  ○ A pkg directory. This directory normally contains three files. Occasionally, there will be more than three, but it depends on the port. Most only require three. The files are:
	  ▪ COMMENT. This is a one-line description of the program.
	  ▪ DESCR. This is a more detailed, often multiple-line, description of the program.
	  ▪ PLIST. This is a list of all the files that will be installed by the port. It also tells the ports system what files to remove upon deinstallation.”[61]
	• “FreeBSD uses a three-stage bootstrap by default, which basically entails three programs which call each other in order (two boot blocks, and the loader). Each of these three build on the previous program's

[60] https://web.archive.org/web/20000815234714/http://www.freebsd.org/handbook/ports.html
[61] https://web.archive.org/web/20000815235607/http://www.freebsd.org/handbook/ports-using.html

’058 Patent Claim 1	Disclosure
	understanding and provide increasing amounts of sophistication. The kernel is then started, which will then probe for devices and initialize them for use. Once the kernel boot process is finished, the kernel passes control to the user process init(8), which then makes sure the disks are in a usable state. init(8) then starts the user-level resource configuration which then mounts filesystems, sets up network cards to act on the network, and generally starts all the processes that usually are run on a FreeBSD system at startup.”[62] • “The easy-to-use command set comprises of: autoboot seconds Proceeds to boot the kernel if not interrupted within the time span given, in seconds. It displays a countdown, and the default timespan is 10 seconds. boot [-options] [kernelname] Immediately proceeds to boot the kernel, with the given options, if any, and with the kernel name given, if it is. boot-conf Goes through the same automatic configuration of modules based on variables as what happens at boot. This only makes sense if you use unload first, and change some variables, most commonly kernel. help [topic] Shows help messages read from /boot/loader.help. If the topic given is index, then the list of available topics is given. include filename ... Processes the file with the given filename. The file is read in, and interpreted line by line. An error immediately stops the include command. load [-t type] filename Loads the kernel, kernel module, or file of the type given, with the filename given. Any arguments after filename are passed to the file. ls [-l] [path] Displays a listing of files in the given path, or the root directory, if the path is not specified. If -l is specified, file sizes will be shown too. lsdev [-v] Lists all of the devices from which it may be possible to load modules. If -v is specified, more details are printed. lsmod [-v] Displays loaded modules. If -v is specified, more details are shown. more filename Display the files specified, with a pause at each LINES displayed.

[62] https://web.archive.org/web/20000815235626/http://www.freebsd.org/handbook/boot.html

’058 Patent Claim 1	Disclosure
	reboot Immediately reboots the system. set variable, set variable=value Set loader's environment variables. unload Removes all loaded modules.”[63] • “Here are some practical examples of loader usage. ○ To simply boot your usual kernel, but in single-user mode: boot -s ○ To unload your usual kernel and modules, and then load just your old (or another) kernel: unload load kernel.old You can use kernel.GENERIC to refer to the generic kernel that comes on the install disk, or kernel.old to refer to your previously installed kernel (when you've upgraded or configured your own kernel, for example). Note: Use the following to load your usual modules with another kernel: unload set kernel=‘kernel.old’ boot-conf ○ To load a kernel configuration script (an automated script which does the things you'd normally do in the kernel boot-time configurator): load -t userconfig_script /boot/kernel.conf”[64] • “5.4. Kernel Interaction During Boot . . . Once the kernel is loaded by either loader (as usual) or boot2 (bypassing the loader), it examines its boot flags, if any, and adjusts its behavior as necessary.”[65] • “5.4.1. Kernel Boot Flags

[63] https://web.archive.org/web/20000815235638/http://www.freebsd.org/handbook/boot-loader.html
[64] https://web.archive.org/web/20000815235638/http://www.freebsd.org/handbook/boot-loader.html
[65] https://web.archive.org/web/20000815235643/http://www.freebsd.org/handbook/boot-kernel.html

’058 Patent Claim 1	Disclosure
	Here are the more common boot flags: -a during kernel initialization, ask for the device to mount as [] the root file system. -C boot from CDROM. -c run UserConfig, the boot-time kernel configurator -s boot into single-user mode -v be more verbose during kernel startup”[66] • “5.5.2. Single-User Mode . . . This mode can be reached through the automatic reboot sequence, or by the user booting with the -s or setting the boot_single variable in loader. It can also be reached by calling shutdown without the reboot (-r) or halt (-h) options, from multi-user mode.If the system console console is set to insecure in /etc/ttys, then the system prompts for the root password before initiating single-user mode. . . . 5.5.3. Multi-User Mode . . . If init finds your filesystems to be in order, or once the user has finished in single-user mode, the system enters multi-user mode, in which it starts the resource configuration of the system. . . . 5.5.3.1. Resource Configuration (rc) . . . The resource configuration system reads in configuration defaults from /etc/defaults/rc.conf, and system-specific details from /etc/rc.conf, and then proceeds to mount the system filesystems mentioned in /etc/fstab, start up networking services, starts up miscellaneous system daemons, and finally runs the startup scripts of locally installed packages. rc(8) is a good reference to the resource configuration system, as is examining the scripts themselves.”[67] • “All access to the system is achieved via accounts, and all processes are run by users, so user and account management are of integral importance on FreeBSD systems. There are three main types of accounts; the Superuser, system users, and user accounts. The Superuser account, usually called root, is used to manage the system with no limitations on privileges. System users run services. Finally, user accounts are used by real people, who log on, read mail, and so forth.”[68] • “The superuser account, usually called root, comes preconfigured, and facilitates system administration, and should not be used for day-to-date tasks like sending and receiving mail, general exploration of the system, or programming. This is because the superuser, unlike normal user accounts, can operate without limits, and misuse of the superuser account may result in spectacular disasters. User accounts are unable to destroy the

[66] https://web.archive.org/web/20000815235643/http://www.freebsd.org/handbook/boot-kernel.html
[67] https://web.archive.org/web/20000815235647/http://www.freebsd.org/handbook/boot-init.html
[68] https://web.archive.org/web/20000815235658/http://www.freebsd.org/handbook/users.html

'058 Patent Claim 1	Disclosure
	system by mistake, so it is generally best to use normal user accounts whenever possible, unless you especially need the extra privilege."[69] • "System users are those used to run services such as DNS, mail, web servers, and so forth. The reason for this is security, as if all services ran as the superuser, they could act without restriction."[70] • "User accounts are the primary means of access for real people to the system, and these accounts insulate the user and the environment, preventing the users from damaging the system or other users, and allowing users to customize their environment without affecting others. Every person accessing your system should have their own unique user account. This allows you to find out who is doing what, and prevent people from clobbering each others' settings, and reading mail meant for the other, and so forth. Each user can set up their own environment to accommodate their use of the system, by using alternate shells, editors, key bindings, and language."[71] • "Almost every device in the kernel has a corresponding 'node' entry in the /dev directory. These nodes look like regular files, but are actually special entries into the kernel which programs use to access the device. The shell script /dev/MAKEDEV, which is executed when you first install the operating system, creates nearly all of the device nodes supported. However, it does not create all of them, so when you add support for a new device, it pays to make sure that the appropriate entries are in this directory, and if not, add them."[72] • "The filesystem is best visualized as a tree, rooted, as it were, at /. /dev, /usr, and the other directories in the root directory are branches, which may have their own branches, such as /usr/local, and so on."[73] • "During the boot process, filesystems listed in /etc/fstab are automatically mounted (unless they are listed with noauto). The /etc/fstab file contains a list of lines of the following format: device /mount-point fstype options dumpfreq passno device is a device name (which should exist), as explained in the Disk naming conventions above. mount-point is a directory (which should exist), on which to mount the filesystem. fstype is the filesystem type to pass to mount(8). The default FreeBSD filesystem is ufs.

[69] https://web.archive.org/web/20000815235703/http://www.freebsd.org/handbook/users-superuser.html
[70] https://web.archive.org/web/20000815235707/http://www.freebsd.org/handbook/users-system.html
[71] https://web.archive.org/web/20000815235712/http://www.freebsd.org/handbook/users-user.html
[72] https://web.archive.org/web/20000815235745/http://www.freebsd.org/handbook/kernelconfig-nodes.html
[73] https://web.archive.org/web/20000815235930/http://www.freebsd.org/handbook/disks-mounting.html

’058 Patent Claim 1	Disclosure
	options is either rw for read-write filesystems, or ro for read-only filesystems, followed by any other options that may be needed. A common option is noauto for filesystems not normally mounted during the boot sequence. Other options in the mount(8) manual page. dumpfreq is the number of days the filesystem should be dumped, and passno is the pass number during which the filesystem is mounted during the boot sequence.”[74] • “The mount(8) command is what is ultimately used to mount filesystems. . . . The umount command takes, as a parameter, one of a mountpoint, a device name, or the -a or -A option. . . . -a and -A are used to unmount all mounted filesystems, possibly modified by the filesystem types listed after -t. -A, however, doesn't attempt to unmount the root filesystem.”[75] • “Quotas are an optional feature of the operating system that allow you to limit the amount of disk space and/or the number of files a user, or members of a group, may allocate on a per-file system basis. This is used most often on timesharing systems where it is desirable to limit the amount of resources any one user or group of users may allocate. This will prevent one user from consuming all of the available disk space. . . . By default the quota files are stored in the root directory of the file system with the names quota.user and quota.group for user and group quotas respectively. See man fstab for more information. Even though that man page says that you can specify an alternate location for the quota files, this is not recommended because the various quota utilities do not seem to handle this properly.”[76] • “A number of daemons ship with the base system that may have problems when run from outside of a jail in a jail-centric environment. This includes syslogd(8), sendmail(8), named(8), and portmap(8). While sendmail and named can be configured to listen only on a specific IP using their configuration files, in most cases it is easier to simply run the daemons in jails only, and not in the host environment. Syslogd cannot be configured to bind only a single IP, but can be configured to not bind a network port, using the ‘-ss’ argument. Attempting to serve NFS from the host environment may also cause confusion, and cannot be easily reconfigured to use only specific IPs, as some NFS services are hosted directly from the kernel. Any third party network software running in the host environment should also be checked and configured so that it does not bind all IP addresses, which would result in those services also appearing to be offered by the jail environments. Once these daemons have been disabled or fixed in the host environment, it is best to reboot so that all daemons are in a known state, to reduce the potential for confusion later (such

[74] https://web.archive.org/web/20000815235930/http://www.freebsd.org/handbook/disks-mounting.html
[75] https://web.archive.org/web/20000815235930/http://www.freebsd.org/handbook/disks-mounting.html
[76] https://web.archive.org/web/20000815235945/http://www.freebsd.org/handbook/quotas.html

'058 Patent Claim 1	Disclosure
	as finding that when you sendmail to a jail, and its sendmail is down, the mail is delivered to the host, etc.)"[77] • "Once a process has been put in a prison, it and its decendants cannot escape the prison. It is not possible to add a process to a preexisting prison. Inside the prison, the concept of 'superuser' is very diluted. In general, it can be assumed that nothing can be mangled from inside a prison which does not exist entirely inside that prison. For instance[,] the directory tree below 'path' can be manipulated all the ways a root can normally do it, including 'rm-rf /*' but new device special nodes cannot be created because they reference shared resources (the device drivers in the kernel)."[78] • "The FreeBSD 'Jail' facility provides the ability to partition the operating system environment, while maintaining the simplicity of the UNIX 'root' model. In Jail, users with privilege find that the scope of their requests is limited to the jail, allowing system administrators to delegate management capabilities for each virtual machine environment. Creating virtual machines in this manner has many potential uses; the most popular thus far has been for providing virtual machine services in Internet Service Provider environments."[79] • "To this end, we describe the new FreeBSD 'Jail' facility, which provides a strong partitioning solution, leveraging existing mechanisms, such as chroot(2), to what effectively amounts to a virtual machine environment. Processes in a jail are provided full access to the files that they may manipulate, processes they may influence, and network services they can make use of, and neither access nor visibility of files, processes or network services outside their partition."[80] • "Jail neatly side-steps the majority of these problems through partitioning. Rather than introduce additional fine-grained access control mechanism, we partition a FreeBSD environment (processes, file system, network resources) into a management environment, and optionally subset Jail environments. . . . Consequently[,] the administrator of a FreeBSD machine can partition the machine into separate jails, and provide access to the super-user account in each of these without losing control of the over-all environment."[81] • "After the creation of the jail tree, the easiest way to configure it is to start up the jail in single-user mode."[82] • "Jail takes advantage of the existing chroot(2) behaviour to limit access to the file system name-space for jailed processes. When a jail is created, it is bound to a particular file system root. Processes are unable to

[77] https://web.archive.org/web/20210506161927/https://www.freebsd.org/cgi/man.cgi?query=jail&apropos=0&sektion=0&manpath=FreeBSD+4.0-RELEASE&arch=default&format=html
[78] https://web.archive.org/web/20220508073711/https://www.freebsd.org/cgi/man.cgi?query=jail&sektion=2&apropos=0&manpath=FreeBSD+4.0-RELEASE
[79] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[80] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[81] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[82] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf

’058 Patent Claim 1	Disclosure
	manipulate files that they cannot address, and as such the integrity and confidentiality of files outside of the jail file system root are protected. Traditional mechanisms for breaking out of chroot(2) have been blocked. In the expected and documented configuration, each jail is provided with its exclusive file system root, and standard FreeBSD directory layout, but this is not mandated by the implementation.”[83] • “While the jail(2) call could be used in a number of ways, the expected configuration creates a complete FreeBSD installation for each jail. This includes copies of all relevant system binaries, data files, and its own /etc directory. Such a configuration maximises the independence of various jails, and reduces the chances of interference between jails being possible, especially when it is desirable to provide root access within a jail to a less trusted user.”[84] • “The jail environments must also be custom-configured. This consists of building and installing a miniature version of the FreeBSD file system tree off of a subdirectory in the host environment, usually /usr/jail, or /data/jail, with a subdirectory per jail. Appropriate instructions for generating this tree are included in the jail(8) man page, but generally this process may be automated using the FreeBSD build environment. One notable difference from the default FreeBSD install is that only a limited set of device nodes should be created. MAKEDEV(8) has been modified to accept a ‘jail’ argument that creates the correct set of nodes. To improve storage efficiency, a fair number of the binaries in the system tree may be deleted, as they are not relevant in a jail environment. This includes the kernel, boot loader, and related files, as well as hardware and network configuration tools.” [85] • “As it stands, the jail code provides a strict subset of system resources to the jail environment, based on access to processes, files, network resources, and privileged services.”[86] • “The jail facility provides FreeBSD with a conceptually simple security partitioning mechanism, allowing the delegation of administrative rights within virtual machine partitions. The implementation relies on restricting access within the jail environment to a well-defined subset of the overall host environment. This includes limiting interaction between processes, and to files, network resources, and privileged operations. Administrative overhead is reduced through avoiding fine-grained access control mechanisms, and maintaining a consistent administrative interface across partitions and the host environment.”[87] • “Your system should have fast access to its root filesystem (/), which contains the kernel and just enough utilities and programs to boot the computer into its most basic running status, single−user mode.”[88]

[83] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[84] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[85] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[86] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[87] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[88] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 25 (2002)

Chart B-10 **Invalidity Contentions: FreeBSD**

’058 Patent Claim 1	Disclosure
	• “You should do everything possible while signed on as a regular user, and only use the root account when you must change the system. That will happen frequently at first, but will grow less common as time passes. Before you can sign on as a regular user, though, you need to set one up for your use.” [89] • “The ‘Home directory’ is where the user's files are kept. The default is generally fine.” [90] • “Whenever your system boots to the point where it can execute userland commands, it runs the shell script /etc/rc. This script mounts all filesystems, brings up the network interfaces, configures device nodes, sets up shared libraries, and does all the other tasks required to set up a system.” • “FreeBSD can run a wide variety of software packages, most of which are available as source code so they can be built as native FreeBSD software. And, thanks to some clever design, FreeBSD can also run software from many foreign operating systems. We'll look at how to do this, focusing on the popular Linux compatibility package that allows FreeBSD to run unmodified Linux software.”[91] • “Managing Shared Libraries . . . The basic idea behind a shared library is quite straightforward: It's a chunk of compiled code that provides services and functions to other chunks of compiled code. Shared libraries provide popular functions for all programs to use, and they are designed to be reused by as many different programs as possible. For example, many programs must hash (or one−way encrypt) data as part of their function. But if every program had to include hashing code, each would be larger, harder to write, and more unpleasant to maintain. What's more, programs would have interoperability problems if they implemented hashes differently. By using a shared library (in this example, libcrypt), a program that needs hashing has access to the functions while eliminating problems of maintenance and interoperability. Similarly, other shared libraries provide common functions to support other software. This reduces the average size of programs, freeing up a reasonably large amount of system memory. FreeBSD builds a cache of available shared libraries at boottime. Programs don't have to scan the whole disk looking for shared libraries; they just ask the cache for the functions they want. In fact, the ability to manage the library cache is one thing that separates a newbie from a professional.”[92] • “When you execute an ELF binary that needs shared libraries, the system calls rtld(1), the ‘run−time linker.’ Rtld examines binaries as they're loaded, determines which shared libraries they need, and loads those libraries. . . . Rather than searching the entire system for anything that looks like a shared library everytime anything is executed, rtld pulls the shared libraries from a library cache. The cache lives on your system in two separate files: /var/run/ld.so.hints (aout) and /var/run/ld−elf.so.hints (ELF). A misconfigured cache is the most likely cause of shared library problems.”[93] • “To see the list of libraries you already have, run ldconfig with the −r flag:

[89] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 31 (2002)
[90] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 214 (2002)
[91] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 250 (2002)
[92] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 252 (2002)
[93] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 253 (2002)

’058 Patent Claim 1	Disclosure
	(Add) ``` fig -r /var/run/ld-elf.so.hints: search directories: /usr/lib:/usr/lib/compat:/usr/X11R6/lib:/usr/local/lib:/usr/local/lib/mysql:/usr/local/pilot/li 0:-lcom_err.2 => /usr/lib/libcom_err.so.2 1:-lscrypt.2 => /usr/lib/libscrypt.so.2 2:-lcrypt.2 => /usr/lib/libcrypt.so.2 ... ``` ldconfig −r examines the shared library cache and lists every shared library it finds. On my system, this list runs to 229 shared libraries. ”[94] • “While the −m option works very well if you're the systems administrator, it won't work if you're just a lowly user without root access. Also, if you have your personal set of shared libraries, your sysadmin won't want to make them globally available, and root must own the shared library directory so that regular users can't just dump things in there willy−nilly. Sysadmins probably won't even want to take the slightest chance of system programs linking against your personal libraries. Here's where the LD_LIBRARY_PATH environment variable appears. Rather than create a cache, LD_LIBRARY_PATH tells the system to check the directories it lists for new shared libraries. . . . You can specify any number of directories in LD_LIBRARY_PATH, separated with colons. For example, I might want to put the directories /home/mwlucas/lib and /compat/linux/usr/lib/local into my LD_LIBRARY_PATH to complete a software install.” [95] • “Lastly, there's the question of what libraries a program expects to have available. You can get this information with ldd(1). This output tells us the names of the shared libraries Emacs requires, and the locations of the files that contain those libraries. You can check this list of required libraries against the output of ldconfig −r to confirm that your program has what it needs. Or you can use this as a shopping list and then go out and get the needed libraries.”[96] • “In addition to recompiling and emulating, the final option for running foreign programs is the one FreeBSD is best known for: *ABI (application binary interface) implementation*. The ABI is the part of the kernel that provides services to programs, including everything from sound−card access to reading files to printing on the screen to starting other programs—all the things a program needs to run. As far as programs are concerned, the ABI is the operating system. By completely implementing the ABI from a different operating

[94] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 253 (2002)
[95] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 254-55 (2002)
[96] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 255 (2002)

’058 Patent Claim 1	Disclosure
	system on your operating system, you can run non−native programs as if they were on the native platform. Or you can provide multiple ABIs and control which ABI a program uses. This is what FreeBSD does.”[97] • “Foreign Software Libraries While the kernel portion of the ABI solves one major issue, the other portions of the system are another problem because every operating system has its own requirements in addition to the kernel. The biggest issue is shared libraries. If the kernel starts a program, and the program can't find its shared libraries, it won't work correctly. No matter which ABI you use, you must have a copy of the shared libraries for that platform. SVR4 and SCO For example, to use the SVR4 and SCO ABIs, you need access to the appropriate system. While a Sun Solaris 2.6 CD will suffice for the SVR4 module, you need to grab the shared libraries from an actual SCO UNIX machine to use the SCO ABI, which means you need a SCO or Solaris license. This isn't an insurmountable problem, of course, but it does make using this module slightly more difficult—and definitely more expensive. OSF/1 A minimal set of OSF/1 shared libraries are available under /usr/ports/emulators/osf1_base. These libraries have a restrictive license and can only be used in fairly narrow circumstances, but you can get a more complete set of shared libraries from an actual OSF/1 system, if you wish. If you have an actual OSF/1 license, you can pretty much do whatever you like with the libraries Linux The shared libraries for the Linux mode are the most freely available of any mode. Since the barrier to entry is so low, we'll discuss Linux compatibility in some detail. Once you have a thorough understanding of how it works, you can apply this knowledge to any other ABI compatibility yo need to implement. If you have a spare Alpha lying around (other than the Multia model known for Random Hea Death), feel free to ship it to me in care of No Starch Press; I'll be delighted to include a discussion of OSF/1 mode in the next edition of Absolute BSD.”[98] • “Alternatively, suppose a Linux binary wants to call sh(1). The Linux ABI will first check under /usr/compat/linux and find bin/sh. When it finds sh there, it will execute that program instead of the

[97] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 257 (2002)
[98] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 259 (2002)

'058 Patent Claim 1	**Disclosure**
	FreeBSD native /bin/sh."[99] • "As I mentioned earlier, the Linux install in linux_base is rather minimal, and some Linux programs expect a broader range of shared libraries to be available. FreeBSD tries to keep ports as small as possible, but compromises by making these additional Linux libraries available as additional ports. The ports. collection includes several ports that augment linux_base, most of which are shared libraries. These increase the range of programs that FreeBSD's Linux mode can support. Installing these ports will round out 99 percent of the functionality you might want to provide to a Linux program. You may need a shared library or program that is not available in linux_base or a port. If so, the simplest thing to do is to find a Linux system of the appropriate version, copy the files, and install them in the appropriate locations under /usr/compat/linux. If you've created a new directory to contain shared libraries, you'll need to tweak /usr/compat/linux/etc/ld.so.conf."[100] • "You'll see a shell prompt, at which point you're in single−user mode in your jail and your jail is up and running."[101] To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant's Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the '058 Patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.
1[d][1] i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software and	FreeBSD, as evidenced by the example citations below, discloses wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications, as this claim limitation appears to be interpreted by VirtaMove. *See, e.g.:* • "FreeBSD has many noteworthy features. Some of these are: . . . *Preemptive multitasking* with dynamic priority adjustment to ensure smooth and fair sharing of the computer between applications and users, even under the heaviest of loads. . . . *Multiuser facilities* which allow many people to use a FreeBSD system simultaneously for a variety of things. This means, for example, that system peripherals such as printers and tape drives are properly shared between all users on the system or the network and that individual resource limits can be placed on users or groups of users, protecting critical system resources from over-use. . . . *Memory protection* ensures that applications (or users) cannot interfere with each other. One application

[99] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 261 (2002)
[100] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 261-62 (2002)
[101] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 180 (2002

’058 Patent Claim 1	Disclosure
	crashing will not affect others in any way. . . .Thousands of *ready-to-run* applications are available from the FreeBSD ports and packages collection. . . . Thousands of additional and *easy-to-port* applications available on the Internet. FreeBSD is source code compatible with most popular commercial Unix systems and thus most applications require few, if any, changes to compile. . . . Demand paged *virtual memory* and ‘merged VM/buffer cache’ design efficiently satisfies applications with large appetites for memory while still maintaining interactive response to other users.”[102] • “Since our release of FreeBSD 2.0 in late 94, the performance, feature set, and stability of FreeBSD has improved dramatically. The largest change is a revamped virtual memory system with a merged VM/file buffer cache that not only increases performance, but also reduces FreeBSD's memory footprint, making a 5MB configuration a more acceptable minimum.”[103] • “In addition to the base distributions, FreeBSD offers a ported software collection with thousands of commonly sought-after programs. By mid-January 2000, there were nearly 3000 ports! The list of ports ranges from http (WWW) servers, to games, languages, editors, and almost everything in between. The entire ports collection requires approximately 50MB of storage, all ports being expressed as ‘deltas’ to their original sources. This makes it much easier for us to update ports, and greatly reduces the disk space demands made by the older 1.0 ports collection. To compile a port, you simply change to the directory of the program you wish to install, type make install, and let the system do the rest. The full original distribution for each port you build is retrieved dynamically off the CDROM or a local FTP site, so you need only enough disk space to build the ports you want. Almost every port is also provided as a pre-compiled ``package'', which can be installed with a simple command (pkg_add) by those who do not wish to compile their own ports from source.”[104] • “FreeBSD, having its history rooted in BSD UNIX, has its fundamentals based on several key UNIX concepts. The first, and most pronounced, is that FreeBSD is a multi-user operating system. The system can handle several users all working simultaneously on completely unrelated tasks. The system is responsible for properly sharing and managing requests for hardware devices, peripherals, memory, and CPU time evenly to each user.”[105] • “Because the system is capable of supporting multiple users, everything the system manages has a set of permissions governing who can read, write, and execute the resource. These permissions are stored as an octet broken into three pieces, one for the owner of the file, one for the group that the file belongs to, and one for everyone else. . . . This is all well and good, but how does the system control permissions on devices? FreeBSD actually treats most hardware devices as a file that programs can open, read, and write data to just like any other file. These special device files are stored on the /dev directory. Directories are also

[102] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html (emphasis in original)
[103] https://web.archive.org/web/20000815230603/http://www.freebsd.org/handbook/history.html
[104] https://web.archive.org/web/20000815230603/http://www.freebsd.org/handbook/history.html
[105] https://web.archive.org/web/20000622040046/http://www.freebsd.org/handbook/permissions.html

'058 Patent Claim 1	Disclosure

	treated as files. They have read, write, and execute permissions. The executable bit for a directory has a slightly different meaning than that of files. When a directory is marked executable, it means it can be searched into, for example, a directory listing can be done in that directory."[106]
	• "Since FreeBSD uses its file systems to determine many fundamental system operations, the hierarchy of the file system is extremely important. Due to the fact that the hier(7) man page provides a complete description of the directory structure, it will not be duplicated here. Please read hier(7) for more information. Of significant importance is the root of all directories, the / directory. This directory is the first directory mounted at boot time and it contains the base system necessary at boot time. The root directory also contains mount points for every other file system that you want to mount. A mount point is a directory where additional file systems can be grafted onto the root file system. Standard mount points include /usr, /var, /mnt, and /cdrom. These directories are usually referenced to entries in the file /etc/fstab. /etc/fstab is a table of various file systems and mount points for reference by the system. Most of the file systems in /etc/fstab are mounted automatically at boot time from the script rc(8) unless they contain the noauto option. Consult the fstab(5) manual page for more information on the format of the /etc/fstab file and the options it contains."[107]
	• "The FreeBSD Ports collection allows you to compile and install a very wide range of applications with a minimum amount of effort. In general, it is a group of skeletons which contain a minimal set of items needed to make an application compile and install cleanly on FreeBSD. Even with all the hype about open standards, getting a program to compile on various UNIX platforms can be a tricky task. Occasionally, you might be lucky enough to find that the program you want compiles cleanly on your system, install everything into all the right directories, and run flawlessly 'out-of-the-box', but this behavior is somewhat rare. Most of the time, you find yourself needing to make modifications in order to get the program to work. This is where the FreeBSD Ports collection comes to the rescue. The general idea behind the Ports collection is to eliminate all of the messy steps involved with making things work properly so that the installation is simple and very painless. With the Ports collection, all of the hard work has already been done for you, and you are able to install any of the Ports collection ports by simply typing make install."[108]
	• "The first thing that should be explained when it comes to the Ports collection is what is actually meant by a 'skeleton'. In a nutshell, a port skeleton is a minimal set of files that are needed for a program to compile and install cleanly on FreeBSD. Each port skeleton includes:
	  ○ A Makefile. The Makefile contains various statements that specify how the application should be compiled and where it should be installed on your system
	  ○ A files directory. The files directory contains a file named md5. This file is named after the MD5 algorithm used to determine ports checksums. A checksum is a number generated by adding up all

[106] https://web.archive.org/web/20000622040046/http://www.freebsd.org/handbook/permissions.html
[107] https://web.archive.org/web/20000621185132/http://www.freebsd.org/handbook/dirstructure.html
[108] https://web.archive.org/web/20000815234714/http://www.freebsd.org/handbook/ports.html

’058 Patent Claim 1	Disclosure
	the data in the file you want to check. If any characters change, the checksum will differ from the original and an error message will be displayed so you are able to investigate the changes. ○ The files directory can also contain other files that are required by the port but do not belong elsewhere in the directory structure. ○ A patches directory. This directory contains patches to make the program compile and install on your FreeBSD system. Patches are basically small files that specify changes to particular files. They are in plain text format, and basically say ‘Remove line 10’ or ‘Change line 26 to this ...’. Patches are also known as ``diffs" because they are generated by the diff program. ○ A pkg directory. This directory normally contains three files. Occasionally, there will be more than three, but it depends on the port. Most only require three. The files are: ▪ COMMENT. This is a one-line description of the program. ▪ DESCR. This is a more detailed, often multiple-line, description of the program. ▪ PLIST. This is a list of all the files that will be installed by the port. It also tells the ports system what files to remove upon deinstallation.”[109] • “FreeBSD uses a three-stage bootstrap by default, which basically entails three programs which call each other in order (two boot blocks, and the loader). Each of these three build on the previous program's understanding and provide increasing amounts of sophistication. The kernel is then started, which will then probe for devices and initialize them for use. Once the kernel boot process is finished, the kernel passes control to the user process init(8), which then makes sure the disks are in a usable state. init(8) then starts the user-level resource configuration which then mounts filesystems, sets up network cards to act on the network, and generally starts all the processes that usually are run on a FreeBSD system at startup.”[110] • “The easy-to-use command set comprises of: autoboot seconds Proceeds to boot the kernel if not interrupted within the time span given, in seconds. It displays a countdown, and the default timespan is 10 seconds. boot [-options] [kernelname] Immediately proceeds to boot the kernel, with the given options, if any, and with the kernel name given, if it is. boot-conf Goes through the same automatic configuration of modules based on variables as what happens at boot. This only makes sense if you use unload first, and change some variables, most commonly kernel. help [topic]

[109] https://web.archive.org/web/20000815235607/http://www.freebsd.org/handbook/ports-using.html
[110] https://web.archive.org/web/20000815235626/http://www.freebsd.org/handbook/boot.html

’058 Patent Claim 1	Disclosure
	Shows help messages read from /boot/loader.help. If the topic given is index, then the list of available topics is given. include filename ... Processes the file with the given filename. The file is read in, and interpreted line by line. An error immediately stops the include command. load [-t type] filename Loads the kernel, kernel module, or file of the type given, with the filename given. Any arguments after filename are passed to the file. ls [-l] [path] Displays a listing of files in the given path, or the root directory, if the path is not specified. If -l is specified, file sizes will be shown too. lsdev [-v] Lists all of the devices from which it may be possible to load modules. If -v is specified, more details are printed. lsmod [-v] Displays loaded modules. If -v is specified, more details are shown. more filename Display the files specified, with a pause at each LINES displayed. reboot Immediately reboots the system. set variable, set variable=value Set loader's environment variables. unload Removes all loaded modules.”[111] • “Here are some practical examples of loader usage. ○ To simply boot your usual kernel, but in single-user mode: boot -s ○ To unload your usual kernel and modules, and then load just your old (or another) kernel: unload load kernel.old You can use kernel.GENERIC to refer to the generic kernel that comes on the install disk, or kernel.old to refer to your previously installed kernel (when you've upgraded or configured your own kernel, for example).

[111] https://web.archive.org/web/20000815235638/http://www.freebsd.org/handbook/boot-loader.html

’058 Patent Claim 1	Disclosure
	Note: Use the following to load your usual modules with another kernel: unload set kernel=‘kernel.old’ boot-conf ○ To load a kernel configuration script (an automated script which does the things you'd normally do in the kernel boot-time configurator): load -t userconfig_script /boot/kernel.conf”[112] • “5.4. Kernel Interaction During Boot . . . Once the kernel is loaded by either loader (as usual) or boot2 (bypassing the loader), it examines its boot flags, if any, and adjusts its behavior as necessary.”[113] • “5.4.1. Kernel Boot Flags Here are the more common boot flags: -a during kernel initialization, ask for the device to mount as [] the root file system. -C boot from CDROM. -c run UserConfig, the boot-time kernel configurator -s boot into single-user mode -v be more verbose during kernel startup”[114] • “5.5.2. Single-User Mode . . . This mode can be reached through the automatic reboot sequence, or by the user booting with the -s or setting the boot_single variable in loader. It can also be reached by calling shutdown without the reboot (-r) or halt (-h) options, from multi-user mode.If the system console console is set to insecure in /etc/ttys, then the system prompts for the root password before initiating single-user mode. . . . 5.5.3. Multi-User Mode . . . If init finds your filesystems to be in order, or once the user has finished in

[112] https://web.archive.org/web/20000815235638/http://www.freebsd.org/handbook/boot-loader.html
[113] https://web.archive.org/web/20000815235643/http://www.freebsd.org/handbook/boot-kernel.html
[114] https://web.archive.org/web/20000815235643/http://www.freebsd.org/handbook/boot-kernel.html

’058 Patent Claim 1	Disclosure
	single-user mode, the system enters multi-user mode, in which it starts the resource configuration of the system. . . . 5.5.3.1. Resource Configuration (rc) . . . The resource configuration system reads in configuration defaults from /etc/defaults/rc.conf, and system-specific details from /etc/rc.conf, and then proceeds to mount the system filesystems mentioned in /etc/fstab, start up networking services, starts up miscellaneous system daemons, and finally runs the startup scripts of locally installed packages. rc(8) is a good reference to the resource configuration system, as is examining the scripts themselves.”[115] • “All access to the system is achieved via accounts, and all processes are run by users, so user and account management are of integral importance on FreeBSD systems. There are three main types of accounts; the Superuser, system users, and user accounts. The Superuser account, usually called root, is used to manage the system with no limitations on privileges. System users run services. Finally, user accounts are used by real people, who log on, read mail, and so forth.”[116] • “The superuser account, usually called root, comes preconfigured, and facilitates system administration, and should not be used for day-to-date tasks like sending and receiving mail, general exploration of the system, or programming. This is because the superuser, unlike normal user accounts, can operate without limits, and misuse of the superuser account may result in spectacular disasters. User accounts are unable to destroy the system by mistake, so it is generally best to use normal user accounts whenever possible, unless you especially need the extra privilege.”[117] • “System users are those used to run services such as DNS, mail, web servers, and so forth. The reason for this is security, as if all services ran as the superuser, they could act without restriction.”[118] • “User accounts are the primary means of access for real people to the system, and these accounts insulate the user and the environment, preventing the users from damaging the system or other users, and allowing users to customize their environment without affecting others. Every person accessing your system should have their own unique user account. This allows you to find out who is doing what, and prevent people from clobbering each others' settings, and reading mail meant for the other, and so forth. Each user can set up their own environment to accommodate their use of the system, by using alternate shells, editors, key bindings, and language.”[119] • “Almost every device in the kernel has a corresponding ‘node’ entry in the /dev directory. These nodes look like regular files, but are actually special entries into the kernel which programs use to access the device. The shell script /dev/MAKEDEV, which is executed when you first install the operating system, creates nearly all of the device nodes supported. However, it does not create all of them, so when you add support

[115] https://web.archive.org/web/20000815235647/http://www.freebsd.org/handbook/boot-init.html
[116] https://web.archive.org/web/20000815235658/http://www.freebsd.org/handbook/users.html
[117] https://web.archive.org/web/20000815235703/http://www.freebsd.org/handbook/users-superuser.html
[118] https://web.archive.org/web/20000815235707/http://www.freebsd.org/handbook/users-system.html
[119] https://web.archive.org/web/20000815235712/http://www.freebsd.org/handbook/users-user.html

’058 Patent Claim 1	Disclosure
	for a new device, it pays to make sure that the appropriate entries are in this directory, and if not, add them.”[120] • “The filesystem is best visualized as a tree, rooted, as it were, at /. /dev, /usr, and the other directories in the root directory are branches, which may have their own branches, such as /usr/local, and so on.”[121] • “During the boot process, filesystems listed in /etc/fstab are automatically mounted (unless they are listed with noauto). The /etc/fstab file contains a list of lines of the following format: device /mount-point fstype options dumpfreq passno device is a device name (which should exist), as explained in the Disk naming conventions above. mount-point is a directory (which should exist), on which to mount the filesystem. fstype is the filesystem type to pass to mount(8). The default FreeBSD filesystem is ufs. options is either rw for read-write filesystems, or ro for read-only filesystems, followed by any other options that may be needed. A common option is noauto for filesystems not normally mounted during the boot sequence. Other options in the mount(8) manual page. dumpfreq is the number of days the filesystem should be dumped, and passno is the pass number during which the filesystem is mounted during the boot sequence.”[122] • “The mount(8) command is what is ultimately used to mount filesystems. . . . The umount command takes, as a parameter, one of a mountpoint, a device name, or the -a or -A option. . . . -a and -A are used to unmount all mounted filesystems, possibly modified by the filesystem types listed after -t. -A, however, doesn't attempt to unmount the root filesystem.”[123] • “jail -- imprison process and its descendants”[124]

[120] https://web.archive.org/web/20000815235745/http://www.freebsd.org/handbook/kernelconfig-nodes.html
[121] https://web.archive.org/web/20000815235930/http://www.freebsd.org/handbook/disks-mounting.html
[122] https://web.archive.org/web/20000815235930/http://www.freebsd.org/handbook/disks-mounting.html
[123] https://web.archive.org/web/20000815235930/http://www.freebsd.org/handbook/disks-mounting.html
[124] https://web.archive.org/web/20210506161927/https://www.freebsd.org/cgi/man.cgi?query=jail&apropos=0&sektion=0&manpath=FreeBSD+4.0-RELEASE&arch=default&format=html

’058 Patent Claim 1	Disclosure
	• “The jail command imprisons a process and all future decendants.”[125] • “A number of daemons ship with the base system that may have problems when run from outside of a jail in a jail-centric environment. This includes syslogd(8), sendmail(8), named(8), and portmap(8). While sendmail and named can be configured to listen only on a specific IP using their configuration files, in most cases it is easier to simply run the daemons in jails only, and not in the host environment. Syslogd cannot be configured to bind only a single IP, but can be configured to not bind a network port, using the ‘-ss’ argument. Attempting to serve NFS from the host environment may also cause confusion, and cannot be easily reconfigured to use only specific IPs, as some NFS services are hosted directly from the kernel. Any third party network software running in the host environment should also be checked and configured so that it does not bind all IP addresses, which would result in those services also appearing to be offered by the jail environments. Once these daemons have been disabled or fixed in the host environment, it is best to reboot so that all daemons are in a known state, to reduce the potential for confusion later (such as finding that when you sendmail to a jail, and its sendmail is down, the mail is delivered to the host, etc.)”[126] • “Once a process has been put in a prison, it and its decendants cannot escape the prison. It is not possible to add a process to a preexisting prison. Inside the prison, the concept of ‘superuser’ is very diluted. In general, it can be assumed that nothing can be mangled from inside a prison which does not exist entirely inside that prison. For instance[,] the directory tree below "path" can be manipulated all the ways a root can normally do it, including ‘rm-rf /*’ but new device special nodes cannot be created because they reference shared resources (the device drivers in the kernel).”[127] • “The FreeBSD ‘Jail’ facility provides the ability to partition the operating system environment, while maintaining the simplicity of the UNIX ‘root’ model. In Jail, users with privilege find that the scope of their requests is limited to the jail, allowing system administrators to delegate management capabilities for each virtual machine environment. Creating virtual machines in this manner has many potential uses; the most popular thus far has been for providing virtual machine services in Internet Service Provider environments.”[128] • “To this end, we describe the new FreeBSD ‘Jail’ facility, which provides a strong partitioning solution, leveraging existing mechanisms, such as chroot(2), to what effectively amounts to a virtual machine environment. Processes in a jail are provided full access to the files that they may manipulate, processes

[125] https://web.archive.org/web/20210506161927/https://www.freebsd.org/cgi/man.cgi?query=jail&apropos=0&sektion=0&manpath=FreeBSD+4.0-RELEASE&arch=default&format=html
[126] https://web.archive.org/web/20210506161927/https://www.freebsd.org/cgi/man.cgi?query=jail&apropos=0&sektion=0&manpath=FreeBSD+4.0-RELEASE&arch=default&format=html
[127] https://web.archive.org/web/20220508073711/https://www.freebsd.org/cgi/man.cgi?query=jail&sektion=2&apropos=0&manpath=FreeBSD+4.0-RELEASE
[128] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf

'058 Patent Claim 1	Disclosure
	they may influence, and network services they can make use of, and neither access nor visibility of files, processes or network services outside their partition."[129] • "Jail neatly side-steps the majority of these problems through partitioning. Rather than introduce additional fine-grained access control mechanism, we partition a FreeBSD environment (processes, file system, network resources) into a management environment, and optionally subset Jail environments. . . . Consequently[,] the administrator of a FreeBSD machine can partition the machine into separate jails, and provide access to the super-user account in each of these without losing control of the over-all environment."[130] • "A process in a partition is referred to as 'in jail'. When a FreeBSD system is booted up after a fresh install, no processes will be in jail. When a process is placed in a jail, it, and any descendents of the process created after the jail creation, will be in that jail. A process may be in only one jail, and after creation, it [cannot] leave the jail. Jails are created when a privileged process calls the jail(2) syscall, with a description of the jail as an argument to the call. Each call to jail(2) creates a new jail; the only way for a new process to enter the jail is by inheriting access to the jail from another process already in that jail. Processes may never leave the jail they created, or were created in." [131] • "Jail takes advantage of the existing chroot(2) behaviour to limit access to the file system name-space for jailed processes. When a jail is created, it is bound to a particular file system root. Processes are unable to manipulate files that they cannot address, and as such the integrity and confidentiality of files outside of the jail file system root are protected. Traditional mechanisms for breaking out of chroot(2) have been blocked. In the expected and documented configuration, each jail is provided with its exclusive file system root, and standard FreeBSD directory layout, but this is not mandated by the implementation."[132] • "While the jail(2) call could be used in a number of ways, the expected configuration creates a complete FreeBSD installation for each jail. This includes copies of all relevant system binaries, data files, and its own /etc directory. Such a configuration maximises the independence of various jails, and reduces the chances of interference between jails being possible, especially when it is desirable to provide root access within a jail to a less trusted user."[133] • "The jail environments must also be custom-configured. This consists of building and installing a miniature version of the FreeBSD file system tree off of a subdirectory in the host environment, usually /usr/jail, or /data/jail, with a subdirectory per jail. Appropriate instructions for generating this tree are included in the jail(8) man page, but generally this process may be automated using the FreeBSD build environment. One notable difference from the default FreeBSD install is that only a limited set of device nodes should be

[129] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[130] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[131] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[132] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[133] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf

’058 Patent Claim 1	Disclosure
	created. MAKEDEV(8) has been modified to accept a ‘jail’ argument that creates the correct set of nodes. To improve storage efficiency, a fair number of the binaries in the system tree may be deleted, as they are not relevant in a jail environment. This includes the kernel, boot loader, and related files, as well as hardware and network configuration tools.” [134] • “As it stands, the jail code provides a strict subset of system resources to the jail environment, based on access to processes, files, network resources, and privileged services.”[135] • “The jail facility provides FreeBSD with a conceptually simple security partitioning mechanism, allowing the delegation of administrative rights within virtual machine partitions. The implementation relies on restricting access within the jail environment to a well-defined subset of the overall host environment. This includes limiting interaction between processes, and to files, network resources, and privileged operations. Administrative overhead is reduced through avoiding fine-grained access control mechanisms, and maintaining a consistent administrative interface across partitions and the host environment.”[136] • “Your system should have fast access to its root filesystem (/), which contains the kernel and just enough utilities and programs to boot the computer into its most basic running status, single−user mode.”[137] • “You should do everything possible while signed on as a regular user, and only use the root account when you must change the system. That will happen frequently at first, but will grow less common as time passes. Before you can sign on as a regular user, though, you need to set one up for your use.” [138] • “The ‘Home directory’ is where the user's files are kept. The default is generally fine.” [139] • “Whenever your system boots to the point where it can execute userland commands, it runs the shell script /etc/rc. This script mounts all filesystems, brings up the network interfaces, configures device nodes, sets up shared libraries, and does all the other tasks required to set up a system.” • “FreeBSD can run a wide variety of software packages, most of which are available as source code so they can be built as native FreeBSD software. And, thanks to some clever design, FreeBSD can also run software from many foreign operating systems. We'll look at how to do this, focusing on the popular Linux compatibility package that allows FreeBSD to run unmodified Linux software.”[140] • “Managing Shared Libraries . . . The basic idea behind a shared library is quite straightforward: It's a chunk of compiled code that provides services and functions to other chunks of compiled code. Shared libraries provide popular functions for all programs to use, and they are designed to be reused by as many different programs as possible. For example, many programs must hash (or one−way encrypt) data as part of their

[134] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[135] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[136] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[137] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* (2002)
[138] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* (2002)
[139] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 214 (2002)
[140] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 250 (2002)

’058 Patent Claim 1	Disclosure

	function. But if every program had to include hashing code, each would be larger, harder to write, and more unpleasant to maintain. What's more, programs would have interoperability problems if they implemented hashes differently. By using a shared library (in this example, libcrypt), a program that needs hashing has access to the functions while eliminating problems of maintenance and interoperability. Similarly, other shared libraries provide common functions to support other software. This reduces the average size of programs, freeing up a reasonably large amount of system memory. FreeBSD builds a cache of available shared libraries at boottime. Programs don't have to scan the whole disk looking for shared libraries; they just ask the cache for the functions they want. In fact, the ability to manage the library cache is one thing that separates a newbie from a professional.”[141]
	• “When you execute an ELF binary that needs shared libraries, the system calls rtld(1), the ‘run−time linker.’ Rtld examines binaries as they're loaded, determines which shared libraries they need, and loads those libraries. . . . Rather than searching the entire system for anything that looks like a shared library everytime anything is executed, rtld pulls the shared libraries from a library cache. The cache lives on your system in two separate files: /var/run/ld.so.hints (aout) and /var/run/ld−elf.so.hints (ELF). A misconfigured cache is the most likely cause of shared library problems.”[142]
	• “To see the list of libraries you already have, run ldconfig with the −r flag:
	(Add)
	`ldconfig -r`
	`/var/run/ld-elf.so.hints:`
	`search directories:`
	`/usr/lib:/usr/lib/compat:/usr/X11R6/lib:/usr/local/lib:/usr/local/lib/mysql:/usr/local/pilot/li`
	`0:-lcom_err.2 => /usr/lib/libcom_err.so.2`
	`1:-lscrypt.2 => /usr/lib/libscrypt.so.2`
	`2:-lcrypt.2 => /usr/lib/libcrypt.so.2`
	`...`
	ldconfig −r examines the shared library cache and lists every shared library it finds. On my system, this list runs to 229 shared libraries. ”[143]
	• “While the −m option works very well if you're the systems administrator, it won't work if you're just a lowly user without root access. Also, if you have your personal set of shared libraries, your sysadmin won't want to make them globally available, and root must own the shared library directory so that regular users can't just dump things in there willy−nilly. Sysadmins probably won't even want to take the slightest chance of system programs linking against your personal libraries. Here's where the LD_LIBRARY_PATH environment variable appears. Rather than create a cache, LD_LIBRARY_PATH tells the system to check

[141] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 252 (2002)
[142] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 253 (2002)
[143] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 253 (2002)

’058 Patent Claim 1	Disclosure
	the directories it lists for new shared libraries. . . . You can specify any number of directories in LD_LIBRARY_PATH, separated with colons. For example, I might want to put the directories /home/mwlucas/lib and /compat/linux/usr/lib/local into my LD_LIBRARY_PATH to complete a software install.” [144] • “Lastly, there's the question of what libraries a program expects to have available. You can get this information with ldd(1). This output tells us the names of the shared libraries Emacs requires, and the locations of the-files that contain those libraries. You can check this list of required libraries against the output of ldconfig −r to confirm that your program has what it needs. Or you can use this as a shopping list and then go out and get the needed libraries.”[145] • “In addition to recompiling and emulating, the final option for running foreign programs is the one FreeBSD is best known for: *ABI (application binary interface) implementation*. The ABI is the part of the kernel that provides services to programs, including everything from sound−card access to reading files to printing on the screen to starting other programs—all the things a program needs to run. As far as programs are concerned, the ABI is the operating system. By completely implementing the ABI from a different operating system on your operating system, you can run non−native programs as if they were on the native platform. Or you can provide multiple ABIs and control which ABI a program uses. This is what FreeBSD does.”[146] • “Foreign Software Libraries While the kernel portion of the ABI solves one major issue, the other portions of the system are another problem because every operating system has its own requirements in addition to the kernel. The biggest issue is shared libraries. If the kernel starts a program, and the program can't find its shared libraries, it won't work correctly. No matter which ABI you use, you must have a copy of the shared libraries for that platform. SVR4 and SCO For example, to use the SVR4 and SCO ABIs, you need access to the appropriate system. While a Sun Solaris 2.6 CD will suffice for the SVR4 module, you need to grab the shared libraries from an actual SCO UNIX machine to use the SCO ABI, which means you need a SCO or Solaris license. This isn't an insurmountable problem, of course, but it does make using this module slightly more difficult—and definitely more expensive. OSF/1

[144] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 254-55 (2002)
[145] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 255 (2002)
[146] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 257 (2002)

’058 Patent Claim 1	Disclosure
	A minimal set of OSF/1 shared libraries are available under /usr/ports/emulators/osf1_base. These libraries have a restrictive license and can only be used in fairly narrow circumstances, but you can get a more complete set of shared libraries from an actual OSF/1 system, if you wish. If you have an actual OSF/1 license, you can pretty much do whatever you like with the libraries Linux The shared libraries for the Linux mode are the most freely available of any mode. Since the barrier to entry is so low, we'll discuss Linux compatibility in some detail. Once you have a thorough understanding of how it works, you can apply this knowledge to any other ABI compatibility yo need to implement. If you have a spare Alpha lying around (other than the Multia model known for Random Hea Death), feel free to ship it to me in care of No Starch Press; I'll be delighted to include a discussion of OSF/1 mode in the next edition of Absolute BSD.”[147] • “Alternatively, suppose a Linux binary wants to call sh(1). The Linux ABI will first check under /usr/compat/linux and find bin/sh. When it finds sh there, it will execute that program instead of the FreeBSD native /bin/sh.”[148] • “As I mentioned earlier, the Linux install in linux_base is rather minimal, and some Linux programs expect a broader range of shared libraries to be available. FreeBSD tries to keep ports as small as possible, but compromises by making these additional Linux libraries available as additional ports. The ports. collection includes several ports that augment linux_base, most of which are shared libraries. These increase the range of programs that FreeBSD's Linux mode can support. Installing these ports will round out 99 percent of the functionality you might want to provide to a Linux program. You may need a shared library or program that is not available in linux_base or a port. If so, the simplest thing to do is to find a Linux system of the appropriate version, copy the files, and install them in the appropriate locations under /usr/compat/linux. If you've created a new directory to contain shared libraries, you'll need to tweak /usr/compat/linux/etc/ld.so.conf.”[149] To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’058 Patent in a manner that would

[147] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 259 (2002)
[148] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 261 (2002)
[149] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 261-62 (2002)

’058 Patent Claim 1	Disclosure
	result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.
1[d][2] when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,	FreeBSD, as evidenced by the example citations below, discloses when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications, as this claim limitation appears to be interpreted by VirtaMove. *See*, *e.g.*, the disclosures set forth at claim element 1[c] and 1[d][1]. This limitation is satisfied by the disclosures set forth at claim elements 1[c] and 1[d][1]. To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’058 Patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.
1[e][1] ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and	FreeBSD, as evidenced by the example citations below, discloses wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, as this claim limitation appears to be interpreted by VirtaMove. *See, e.g.*, the disclosures set forth at claim element 1[c]. *See also*, *e.g.:* • “FreeBSD has many noteworthy features. Some of these are: . . . *Preemptive multitasking* with dynamic priority adjustment to ensure smooth and fair sharing of the computer between applications and users, even under the heaviest of loads. . . . *Multiuser facilities* which allow many people to use a FreeBSD system simultaneously for a variety of things. This means, for example, that system peripherals such as printers and tape drives are properly shared between all users on the system or the network and that individual resource limits can be placed on users or groups of users, protecting critical system resources from over-use. . . . *Memory protection* ensures that applications (or users) cannot interfere with each other. One application crashing will not affect others in any way. . . .Thousands of *ready-to-run* applications are available from the FreeBSD ports and packages collection. . . . Thousands of additional and *easy-to-port* applications available on the Internet. FreeBSD is source code compatible with most popular commercial Unix systems and thus most applications require few, if any, changes to compile. . . . Demand paged *virtual memory* and ‘merged

’058 Patent Claim 1	Disclosure
	VM/buffer cache’ design efficiently satisfies applications with large appetites for memory while still maintaining interactive response to other users.”[150] • “Since our release of FreeBSD 2.0 in late 94, the performance, feature set, and stability of FreeBSD has improved dramatically. The largest change is a revamped virtual memory system with a merged VM/file buffer cache that not only increases performance, but also reduces FreeBSD's memory footprint, making a 5MB configuration a more acceptable minimum.”[151] • “In addition to the base distributions, FreeBSD offers a ported software collection with thousands of commonly sought-after programs. By mid-January 2000, there were nearly 3000 ports! The list of ports ranges from http (WWW) servers, to games, languages, editors, and almost everything in between. The entire ports collection requires approximately 50MB of storage, all ports being expressed as ‘deltas’ to their original sources. This makes it much easier for us to update ports, and greatly reduces the disk space demands made by the older 1.0 ports collection. To compile a port, you simply change to the directory of the program you wish to install, type make install, and let the system do the rest. The full original distribution for each port you build is retrieved dynamically off the CDROM or a local FTP site, so you need only enough disk space to build the ports you want. Almost every port is also provided as a pre-compiled ‘package’, which can be installed with a simple command (pkg_add) by those who do not wish to compile their own ports from source.”[152] • “FreeBSD, having its history rooted in BSD UNIX, has its fundamentals based on several key UNIX concepts. The first, and most pronounced, is that FreeBSD is a multi-user operating system. The system can handle several users all working simultaneously on completely unrelated tasks. The system is responsible for properly sharing and managing requests for hardware devices, peripherals, memory, and CPU time evenly to each user.”[153] • “Because the system is capable of supporting multiple users, everything the system manages has a set of permissions governing who can read, write, and execute the resource. These permissions are stored as an octet broken into three pieces, one for the owner of the file, one for the group that the file belongs to, and one for everyone else. . . . This is all well and good, but how does the system control permissions on devices? FreeBSD actually treats most hardware devices as a file that programs can open, read, and write data to just like any other file. These special device files are stored on the /dev directory. Directories are also treated as files. They have read, write, and execute permissions. The executable bit for a directory has a slightly different meaning than that of files. When a directory is marked executable, it means it can be searched into, for example, a directory listing can be done in that directory.”[154]

[150] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html (emphasis in original)
[151] https://web.archive.org/web/20000815230603/http://www.freebsd.org/handbook/history.html
[152] https://web.archive.org/web/20000815230603/http://www.freebsd.org/handbook/history.html
[153] https://web.archive.org/web/20000622040046/http://www.freebsd.org/handbook/permissions.html
[154] https://web.archive.org/web/20000622040046/http://www.freebsd.org/handbook/permissions.html

’058 Patent Claim 1	Disclosure

	• “Since FreeBSD uses its file systems to determine many fundamental system operations, the hierarchy of the file system is extremely important. Due to the fact that the hier(7) man page provides a complete description of the directory structure, it will not be duplicated here. Please read hier(7) for more information. Of significant importance is the root of all directories, the / directory. This directory is the first directory mounted at boot time and it contains the base system necessary at boot time. The root directory also contains mount points for every other file system that you want to mount. A mount point is a directory where additional file systems can be grafted onto the root file system. Standard mount points include /usr, /var, /mnt, and /cdrom. These directories are usually referenced to entries in the file /etc/fstab. /etc/fstab is a table of various file systems and mount points for reference by the system. Most of the file systems in /etc/fstab are mounted automatically at boot time from the script rc(8) unless they contain the noauto option. Consult the fstab(5) manual page for more information on the format of the /etc/fstab file and the options it contains.”[155]
	• “The FreeBSD Ports collection allows you to compile and install a very wide range of applications with a minimum amount of effort. In general, it is a group of skeletons which contain a minimal set of items needed to make an application compile and install cleanly on FreeBSD. Even with all the hype about open standards, getting a program to compile on various UNIX platforms can be a tricky task. Occasionally, you might be lucky enough to find that the program you want compiles cleanly on your system, install everything into all the right directories, and run flawlessly ‘out-of-the-box’, but this behavior is somewhat rare. Most of the time, you find yourself needing to make modifications in order to get the program to work. This is where the FreeBSD Ports collection comes to the rescue. The general idea behind the Ports collection is to eliminate all of the messy steps involved with making things work properly so that the installation is simple and very painless. With the Ports collection, all of the hard work has already been done for you, and you are able to install any of the Ports collection ports by simply typing make install.”[156]
	• “The first thing that should be explained when it comes to the Ports collection is what is actually meant by a ‘skeleton’. In a nutshell, a port skeleton is a minimal set of files that are needed for a program to compile and install cleanly on FreeBSD. Each port skeleton includes:
	  ○ A Makefile. The Makefile contains various statements that specify how the application should be compiled and where it should be installed on your system
	  ○ A files directory. The files directory contains a file named md5. This file is named after the MD5 algorithm used to determine ports checksums. A checksum is a number generated by adding up all the data in the file you want to check. If any characters change, the checksum will differ from the original and an error message will be displayed so you are able to investigate the changes.
	  ○ The files directory can also contain other files that are required by the port but do not belong elsewhere in the directory structure.

[155] https://web.archive.org/web/20000621185132/http://www.freebsd.org/handbook/dirstructure.html
[156] https://web.archive.org/web/20000815234714/http://www.freebsd.org/handbook/ports.html

’058 Patent Claim 1	Disclosure
	○ A patches directory. This directory contains patches to make the program compile and install on your FreeBSD system. Patches are basically small files that specify changes to particular files. They are in plain text format, and basically say ‘Remove line 10’ or ‘Change line 26 to this ...’. Patches are also known as ``diffs'' because they are generated by the diff program. ○ A pkg directory. This directory normally contains three files. Occasionally, there will be more than three, but it depends on the port. Most only require three. The files are: ▪ COMMENT. This is a one-line description of the program. ▪ DESCR. This is a more detailed, often multiple-line, description of the program. ▪ PLIST. This is a list of all the files that will be installed by the port. It also tells the ports system what files to remove upon deinstallation.”[157] • “FreeBSD uses a three-stage bootstrap by default, which basically entails three programs which call each other in order (two boot blocks, and the loader). Each of these three build on the previous program's understanding and provide increasing amounts of sophistication. The kernel is then started, which will then probe for devices and initialize them for use. Once the kernel boot process is finished, the kernel passes control to the user process init(8), which then makes sure the disks are in a usable state. init(8) then starts the user-level resource configuration which then mounts filesystems, sets up network cards to act on the network, and generally starts all the processes that usually are run on a FreeBSD system at startup.”[158] • “The easy-to-use command set comprises of: autoboot seconds Proceeds to boot the kernel if not interrupted within the time span given, in seconds. It displays a countdown, and the default timespan is 10 seconds. boot [-options] [kernelname] Immediately proceeds to boot the kernel, with the given options, if any, and with the kernel name given, if it is. boot-conf Goes through the same automatic configuration of modules based on variables as what happens at boot. This only makes sense if you use unload first, and change some variables, most commonly kernel. help [topic] Shows help messages read from /boot/loader.help. If the topic given is index, then the list of available topics is given. include filename ... Processes the file with the given filename. The file is read in, and interpreted line by line. An error immediately stops the include command.

[157] https://web.archive.org/web/20000815235607/http://www.freebsd.org/handbook/ports-using.html
[158] https://web.archive.org/web/20000815235626/http://www.freebsd.org/handbook/boot.html

’058 Patent Claim 1	Disclosure
	load [-t type] filename Loads the kernel, kernel module, or file of the type given, with the filename given. Any arguments after filename are passed to the file. ls [-l] [path] Displays a listing of files in the given path, or the root directory, if the path is not specified. If -l is specified, file sizes will be shown too. lsdev [-v] Lists all of the devices from which it may be possible to load modules. If -v is specified, more details are printed. lsmod [-v] Displays loaded modules. If -v is specified, more details are shown. more filename Display the files specified, with a pause at each LINES displayed. reboot Immediately reboots the system. set variable, set variable=value Set loader's environment variables. unload Removes all loaded modules.”[159] • “Here are some practical examples of loader usage. ○ To simply boot your usual kernel, but in single-user mode: boot -s ○ To unload your usual kernel and modules, and then load just your old (or another) kernel: unload load kernel.old You can use kernel.GENERIC to refer to the generic kernel that comes on the install disk, or kernel.old to refer to your previously installed kernel (when you've upgraded or configured your own kernel, for example). Note: Use the following to load your usual modules with another kernel: unload set kernel=‘kernel.old’

[159] https://web.archive.org/web/20000815235638/http://www.freebsd.org/handbook/boot-loader.html

'058 Patent Claim 1	Disclosure
	boot-conf o To load a kernel configuration script (an automated script which does the things you'd normally do in the kernel boot-time configurator): load -t userconfig_script /boot/kernel.conf"[160] • "5.4. Kernel Interaction During Boot . . . Once the kernel is loaded by either loader (as usual) or boot2 (bypassing the loader), it examines its boot flags, if any, and adjusts its behavior as necessary."[161] • "5.4.1. Kernel Boot Flags Here are the more common boot flags: -a during kernel initialization, ask for the device to mount as [] the root file system. -C boot from CDROM. -c run UserConfig, the boot-time kernel configurator -s boot into single-user mode -v be more verbose during kernel startup"[162] • "5.5.2. Single-User Mode . . . This mode can be reached through the automatic reboot sequence, or by the user booting with the -s or setting the boot_single variable in loader. It can also be reached by calling shutdown without the reboot (-r) or halt (-h) options, from multi-user mode.If the system console console is set to insecure in /etc/ttys, then the system prompts for the root password before initiating single-user mode. . . . 5.5.3. Multi-User Mode . . . If init finds your filesystems to be in order, or once the user has finished in single-user mode, the system enters multi-user mode, in which it starts the resource configuration of the system. . . . 5.5.3.1. Resource Configuration (rc) . . . The resource configuration system reads in configuration defaults from /etc/defaults/rc.conf, and system-specific details from /etc/rc.conf, and then proceeds to mount the system filesystems mentioned in /etc/fstab, start up networking services, starts up

[160] https://web.archive.org/web/20000815235638/http://www.freebsd.org/handbook/boot-loader.html
[161] https://web.archive.org/web/20000815235643/http://www.freebsd.org/handbook/boot-kernel.html
[162] https://web.archive.org/web/20000815235643/http://www.freebsd.org/handbook/boot-kernel.html

’058 Patent Claim 1	Disclosure
	miscellaneous system daemons, and finally runs the startup scripts of locally installed packages. rc(8) is a good reference to the resource configuration system, as is examining the scripts themselves.”[163] • “All access to the system is achieved via accounts, and all processes are run by users, so user and account management are of integral importance on FreeBSD systems. There are three main types of accounts; the Superuser, system users, and user accounts. The Superuser account, usually called root, is used to manage the system with no limitations on privileges. System users run services. Finally, user accounts are used by real people, who log on, read mail, and so forth.”[164] • “The superuser account, usually called root, comes preconfigured, and facilitates system administration, and should not be used for day-to-date tasks like sending and receiving mail, general exploration of the system, or programming. This is because the superuser, unlike normal user accounts, can operate without limits, and misuse of the superuser account may result in spectacular disasters. User accounts are unable to destroy the system by mistake, so it is generally best to use normal user accounts whenever possible, unless you especially need the extra privilege.”[165] • “System users are those used to run services such as DNS, mail, web servers, and so forth. The reason for this is security, as if all services ran as the superuser, they could act without restriction.”[166] • “User accounts are the primary means of access for real people to the system, and these accounts insulate the user and the environment, preventing the users from damaging the system or other users, and allowing users to customize their environment without affecting others. Every person accessing your system should have their own unique user account. This allows you to find out who is doing what, and prevent people from clobbering each others' settings, and reading mail meant for the other, and so forth. Each user can set up their own environment to accommodate their use of the system, by using alternate shells, editors, key bindings, and language.”[167] • “Almost every device in the kernel has a corresponding ‘node’ entry in the /dev directory. These nodes look like regular files, but are actually special entries into the kernel which programs use to access the device. The shell script /dev/MAKEDEV, which is executed when you first install the operating system, creates nearly all of the device nodes supported. However, it does not create all of them, so when you add support for a new device, it pays to make sure that the appropriate entries are in this directory, and if not, add them.”[168]

[163] https://web.archive.org/web/20000815235647/http://www.freebsd.org/handbook/boot-init.html
[164] https://web.archive.org/web/20000815235658/http://www.freebsd.org/handbook/users.html
[165] https://web.archive.org/web/20000815235703/http://www.freebsd.org/handbook/users-superuser.html
[166] https://web.archive.org/web/20000815235707/http://www.freebsd.org/handbook/users-system.html
[167] https://web.archive.org/web/20000815235712/http://www.freebsd.org/handbook/users-user.html
[168] https://web.archive.org/web/20000815235745/http://www.freebsd.org/handbook/kernelconfig-nodes.html

’058 Patent Claim 1	Disclosure
	• “The filesystem is best visualized as a tree, rooted, as it were, at /. /dev, /usr, and the other directories in the root directory are branches, which may have their own branches, such as /usr/local, and so on.”[169] • “During the boot process, filesystems listed in /etc/fstab are automatically mounted (unless they are listed with noauto). The /etc/fstab file contains a list of lines of the following format: device /mount-point fstype options dumpfreq passno device is a device name (which should exist), as explained in the Disk naming conventions above. mount-point is a directory (which should exist), on which to mount the filesystem. fstype is the filesystem type to pass to mount(8). The default FreeBSD filesystem is ufs. options is either rw for read-write filesystems, or ro for read-only filesystems, followed by any other options that may be needed. A common option is noauto for filesystems not normally mounted during the boot sequence. Other options in the mount(8) manual page. dumpfreq is the number of days the filesystem should be dumped, and passno is the pass number during which the filesystem is mounted during the boot sequence.”[170] • “The mount(8) command is what is ultimately used to mount filesystems. . . . The umount command takes, as a parameter, one of a mountpoint, a device name, or the -a or -A option. . . . -a and -A are used to unmount all mounted filesystems, possibly modified by the filesystem types listed after -t. -A, however, doesn't attempt to unmount the root filesystem.”[171] • “jail -- imprison process and its descendants”[172] • “The jail command imprisons a process and all future decendants.”[173] • “A number of daemons ship with the base system that may have problems when run from outside of a jail in a jail-centric environment. This includes syslogd(8), sendmail(8), named(8), and portmap(8). While

[169] https://web.archive.org/web/20000815235930/http://www.freebsd.org/handbook/disks-mounting.html
[170] https://web.archive.org/web/20000815235930/http://www.freebsd.org/handbook/disks-mounting.html
[171] https://web.archive.org/web/20000815235930/http://www.freebsd.org/handbook/disks-mounting.html
[172] https://web.archive.org/web/20210506161927/https://www.freebsd.org/cgi/man.cgi?query=jail&apropos=0&sektion=0&manpath=FreeBSD+4.0-RELEASE&arch=default&format=html
[173] https://web.archive.org/web/20210506161927/https://www.freebsd.org/cgi/man.cgi?query=jail&apropos=0&sektion=0&manpath=FreeBSD+4.0-RELEASE&arch=default&format=html

Chart B-10 **Invalidity Contentions: FreeBSD**

’058 Patent Claim 1	Disclosure
	sendmail and named can be configured to listen only on a specific IP using their configuration files, in most cases it is easier to simply run the daemons in jails only, and not in the host environment. Syslogd cannot be configured to bind only a single IP, but can be configured to not bind a network port, using the ``-ss" argument. Attempting to serve NFS from the host environment may also cause confusion, and cannot be easily reconfigured to use only specific IPs, as some NFS services are hosted directly from the kernel. Any third party network software running in the host environment should also be checked and configured so that it does not bind all IP addresses, which would result in those services also appearing to be offered by the jail environments. Once these daemons have been disabled or fixed in the host environment, it is best to reboot so that all daemons are in a known state, to reduce the potential for confusion later (such as finding that when you sendmail to a jail, and its sendmail is down, the mail is delivered to the host, etc.)”[174] • “Once a process has been put in a prison, it and its decendants cannot escape the prison. It is not possible to add a process to a preexisting prison. Inside the prison, the concept of "superuser" is very diluted. In general, it can be assumed that nothing can be mangled from inside a prison which does not exist entirely inside that prison. For instance[,] the directory tree below "path" can be manipulated all the ways a root can normally do it, including "rm-rf /*" but new device special nodes cannot be created because they reference shared resources (the device drivers in the kernel).”[175] • “The FreeBSD ‘Jail’ facility provides the ability to partition the operating system environment, while maintaining the simplicity of the UNIX ‘‘root’’ model. In Jail, users with privilege find that the scope of their requests is limited to the jail, allowing system administrators to delegate management capabilities for each virtual machine environment. Creating virtual machines in this manner has many potential uses; the most popular thus far has been for providing virtual machine services in Internet Service Provider environments.”[176] • “To this end, we describe the new FreeBSD ‘Jail’ facility, which provides a strong partitioning solution, leveraging existing mechanisms, such as chroot(2), to what effectively amounts to a virtual machine environment. Processes in a jail are provided full access to the files that they may manipulate, processes they may influence, and network services they can make use of, and neither access nor visibility of files, processes or network services outside their partition.”[177] • “Jail neatly side-steps the majority of these problems through partitioning. Rather than introduce additional fine-grained access control mechanism, we partition a FreeBSD environment (processes, file system, network resources) into a management environment, and optionally subset Jail environments. . . .

[174] https://web.archive.org/web/20210506161927/https://www.freebsd.org/cgi/man.cgi?query=jail&apropos=0&sektion=0&manpath=FreeBSD+4.0-RELEASE&arch=default&format=html
[175] https://web.archive.org/web/20220508073711/https://www.freebsd.org/cgi/man.cgi?query=jail&sektion=2&apropos=0&manpath=FreeBSD+4.0-RELEASE
[176] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[177] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf

'058 Patent Claim 1	Disclosure
	Consequently[,] the administrator of a FreeBSD machine can partition the machine into separate jails, and provide access to the super-user account in each of these without losing control of the over-all environment."[178] • "A process in a partition is referred to as 'in jail'. When a FreeBSD system is booted up after a fresh install, no processes will be in jail. When a process is placed in a jail, it, and any descendents of the process created after the jail creation, will be in that jail. A process may be in only one jail, and after creation, it [cannot] leave the jail. Jails are created when a privileged process calls the jail(2) syscall, with a description of the jail as an argument to the call. Each call to jail(2) creates a new jail; the only way for a new process to enter the jail is by inheriting access to the jail from another process already in that jail. Processes may never leave the jail they created, or were created in." [179] • "Jail takes advantage of the existing chroot(2) behaviour to limit access to the file system name-space for jailed processes. When a jail is created, it is bound to a particular file system root. Processes are unable to manipulate files that they cannot address, and as such the integrity and confidentiality of files outside of the jail file system root are protected. Traditional mechanisms for breaking out of chroot(2) have been blocked. In the expected and documented configuration, each jail is provided with its exclusive file system root, and standard FreeBSD directory layout, but this is not mandated by the implementation."[180] • "Processes running without root privileges will notice few, if any differences between a jailed environment or un-jailed environment. Processes running with root privileges will find that many restrictions apply to the privileged calls they may make. Some calls will now return an access error — for example, an attempt to create a device node will now fail. Others will have a more limited scope than normal — attempts to bind a reserved port number on all available addresses will result in binding only the address associated with the jail. Other calls will succeed as normal: root may read a file owned by any uid, as long as it is accessible through the jail file system name-space. Processes within the jail will find that they are unable to interact or even verify the existence of processes outside the jail — processes within the jail are prevented from delivering signals to processes outside the jail, as well as connecting to those processes with debuggers, or even see them in the sysctl or process file system monitoring mechanisms. Jail does not prevent, nor is it intended to prevent, the use of covert channels or communications mechanisms via accepted interfaces — for example, two processes may communicate via sockets over the IP network interface. Nor does it attempt to provide scheduling services based on the partition; however, it does prevent calls that interfere with normal process operation."[181] • "Processes running with root privileges in the jail find that there are serious restrictions on what it is capable of doing — in particular, activities that would extend outside of the jail:

[178] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[179] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[180] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[181] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf

’058 Patent Claim 1	Disclosure
	○ Modifying the running kernel by direct access and loading kernel modules is prohibited. ○ Modifying any of the network configuration, interfaces, addresses, and routing table is prohibited. ○ Mounting and unmounting file systems is prohibited. ○ Creating device nodes is prohibited. ○ Accessing raw, divert, or routing sockets is prohibited. ○ Modifying kernel runtime parameters, such as most sysctl settings, is prohibited. ○ Changing secure level-related file flags is prohibited. ○ Accessing network resources not associated with the jail is prohibited. . . . These restrictions on root access limit the scope of root processes, enabling most applications to run un-hindered, but preventing calls that might allow an application to reach beyond the jail and influence other processes or system-wide configuration”[182] • “While the jail(2) call could be used in a number of ways, the expected configuration creates a complete FreeBSD installation for each jail. This includes copies of all relevant system binaries, data files, and its own /etc directory. Such a configuration maximises the independence of various jails, and reduces the chances of interference between jails being possible, especially when it is desirable to provide root access within a jail to a less trusted user.”[183] • “A few warnings are generated for sysctl’s that are not permitted to be set within the jail, but the end result is a set of processes in an isolated process environment, bound to a single IP address” [184] • “One aspect immediately observable in an environment with multiple jails is that uids and gids are local to each jail environment: the uid associated with a process in one jail may be for a different user than in another jail. This collision of identifiers is only visible in the host environment, as normally processes from one jail are never visible in an environment with another scope for user/uid and group/gid mapping. Managers in the host environment should understand these scoping issues, or confusion and unintended consequences may result.” [185] • “The jail facility provides FreeBSD with a conceptually simple security partitioning mechanism, allowing the delegation of administrative rights within virtual machine partitions. The implementation relies on restricting access within the jail environment to a well-defined subset of the overall host environment. This includes limiting interaction between processes, and to files, network resources, and privileged operations. Administrative overhead is reduced through avoiding fine-grained access control mechanisms, and maintaining a consistent administrative interface across partitions and the host environment.”[186]

[182] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[183] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[184] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[185] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[186] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf

’058 Patent Claim 1	Disclosure
	• “Your system should have fast access to its root filesystem (/), which contains the kernel and just enough utilities and programs to boot the computer into its most basic running status, single−user mode.”[187] • “You should do everything possible while signed on as a regular user, and only use the root account when you must change the system. That will happen frequently at first, but will grow less common as time passes. Before you can sign on as a regular user, though, you need to set one up for your use.” [188] • “The ‘Home directory’ is where the user's files are kept. The default is generally fine.” [189] • “Whenever your system boots to the point where it can execute userland commands, it runs the shell script /etc/rc. This script mounts all filesystems, brings up the network interfaces, configures device nodes, sets up shared libraries, and does all the other tasks required to set up a system.” • “FreeBSD can run a wide variety of software packages, most of which are available as source code so they can be built as native FreeBSD software. And, thanks to some clever design, FreeBSD can also run software from many foreign operating systems. We'll look at how to do this, focusing on the popular Linux compatibility package that allows FreeBSD to run unmodified Linux software.”[190] • “Managing Shared Libraries . . . The basic idea behind a shared library is quite straightforward: It's a chunk of compiled code that provides services and functions to other chunks of compiled code. Shared libraries provide popular functions for all programs to use, and they are designed to be reused by as many different programs as possible. For example, many programs must hash (or one−way encrypt) data as part of their function. But if every program had to include hashing code, each would be larger, harder to write, and more unpleasant to maintain. What's more, programs would have interoperability problems if they implemented hashes differently. By using a shared library (in this example, libcrypt), a program that needs hashing has access to the functions while eliminating problems of maintenance and interoperability. Similarly, other shared libraries provide common functions to support other software. This reduces the average size of programs, freeing up a reasonably large amount of system memory. FreeBSD builds a cache of available shared libraries at boottime. Programs don't have to scan the whole disk looking for shared libraries; they just ask the cache for the functions they want. In fact, the ability to manage the library cache is one thing that separates a newbie from a professional.”[191] • “When you execute an ELF binary that needs shared libraries, the system calls rtld(1), the ‘run−time linker.’ Rtld examines binaries as they're loaded, determines which shared libraries they need, and loads those libraries. . . . Rather than searching the entire system for anything that looks like a shared library everytime anything is executed, rtld pulls the shared libraries from a library cache. The cache lives on your system in

[187] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* (2002)
[188] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* (2002)
[189] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 214 (2002)
[190] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 250 (2002)
[191] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 252 (2002)

’058 Patent Claim 1	Disclosure
	two separate files: /var/run/ld.so.hints (aout) and /var/run/ld−elf.so.hints (ELF). A misconfigured cache is the most likely cause of shared library problems.”[192] • “To see the list of libraries you already have, run ldconfig with the −r flag:  ldconfig −r examines the shared library cache and lists every shared library it finds. On my system, this list runs to 229 shared libraries. ”[193] • “While the −m option works very well if you're the systems administrator, it won't work if you're just a lowly user without root access. Also, if you have your personal set of shared libraries, your sysadmin won't want to make them globally available, and root must own the shared library directory so that regular users can't just dump things in there willy−nilly. Sysadmins probably won't even want to take the slightest chance of system programs linking against your personal libraries. Here's where the LD_LIBRARY_PATH environment variable appears. Rather than create a cache, LD_LIBRARY_PATH tells the system to check the directories it lists for new shared libraries. . . . You can specify any number of directories in LD_LIBRARY_PATH, separated with colons. For example, I might want to put the directories /home/mwlucas/lib and /compat/linux/usr/lib/local into my LD_LIBRARY_PATH to complete a software install.” [194] • “Lastly, there's the question of what libraries a program expects to have available. You can get this information with ldd(1). This output tells us the names of the shared libraries Emacs requires, and the locations of-the files that contain those libraries. You can check this list of required libraries against the output of ldconfig −r to confirm that your program has what it needs. Or you can use this as a shopping list and then go out and get the needed libraries.”[195] • “In addition to recompiling and emulating, the final option for running foreign programs is the one FreeBSD is best known for: *ABI (application binary interface) implementation*. The ABI is the part of the kernel that provides services to programs, including everything from sound−card access to reading files to printing on

[192] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 253 (2002)
[193] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 253 (2002)
[194] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 254-55 (2002)
[195] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 255 (2002)

’058 Patent Claim 1	Disclosure
	the screen to starting other programs—all the things a program needs to run. As far as programs are concerned, the ABI is the operating system. By completely implementing the ABI from a different operating system on your operating system, you can run non−native programs as if they were on the native platform. Or you can provide multiple ABIs and control which ABI a program uses. This is what FreeBSD does.”[196] • “Foreign Software Libraries While the kernel portion of the ABI solves one major issue, the other portions of the system are another problem because every operating system has its own requirements in addition to the kernel. The biggest issue is shared libraries. If the kernel starts a program, and the program can't find its shared libraries, it won't work correctly. No matter which ABI you use, you must have a copy of the shared libraries for that platform. SVR4 and SCO For example, to use the SVR4 and SCO ABIs, you need access to the appropriate system. While a Sun Solaris 2.6 CD will suffice for the SVR4 module, you need to grab the shared libraries from an actual SCO UNIX machine to use the SCO ABI, which means you need a SCO or Solaris license. This isn't an insurmountable problem, of course, but it does make using this module slightly more difficult—and definitely more expensive. OSF/1 A minimal set of OSF/1 shared libraries are available under /usr/ports/emulators/osf1_base. These libraries have a restrictive license and can only be used in fairly narrow circumstances, but you can get a more complete set of shared libraries from an actual OSF/1 system, if you wish. If you have an actual OSF/1 license, you can pretty much do whatever you like with the libraries Linux The shared libraries for the Linux mode are the most freely available of any mode. Since the barrier to entry is so low, we'll discuss Linux compatibility in some detail. Once you have a thorough understanding of how it works, you can apply this knowledge to any other ABI compatibility you need to implement. If you have a spare Alpha lying around (other than the Multia model known for Random Hea Death), feel free to ship it to me in care of No Starch Press; I'll be delighted to include a discussion of OSF/1 mode in the next edition of Absolute BSD.”[197]

[196] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 257 (2002)
[197] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 259 (2002)

Chart B-10 **Invalidity Contentions: FreeBSD**

’058 Patent Claim 1	Disclosure
	• “Alternatively, suppose a Linux binary wants to call sh(1). The Linux ABI will first check under /usr/compat/linux and find bin/sh. When it finds sh there, it will execute that program instead of the FreeBSD native /bin/sh.”[198] • “As I mentioned earlier, the Linux install in linux_base is rather minimal, and some Linux programs expect a broader range of shared libraries to be available. FreeBSD tries to keep ports as small as possible, but compromises by making these additional Linux libraries available as additional ports. The ports. collection includes several ports that augment linux_base, most of which are shared libraries. These increase the range of programs that FreeBSD's Linux mode can support. Installing these ports will round out 99 percent of the functionality you might want to provide to a Linux program. You may need a shared library or program that is not available in linux_base or a port. If so, the simplest thing to do is to find a Linux system of the appropriate version, copy the files, and install them in the appropriate locations under /usr/compat/linux. If you've created a new directory to contain shared libraries, you'll need to tweak /usr/compat/linux/etc/ld.so.conf.”[199] • “Unless you're on FreeBSD. FreeBSD administrators faced this problem long ago, and solved it by improving the chroot process dramatically. In fact, they solved it so well that, when using FreeBSD, you can build an entire virtual machine on disk, and isolate that machine from the rest of your system. This is called a jail.”[200] • “Think of a jail as something like a client−server environment. The main server is the host system ,and each jailed system is a client. Changes made to the host can be reflected across all systems, but changes to the jail can't affect the main system, unless you allow a jail to fill up a disk drive or some such. When in jail, clients can have root access and even install whatever nifty toys they desire without interfering with the main system. All processes that are running in the jail are restricted to the jail environment, and the kernel does not give them access to any information not in their jail. The filesystem in the jail does not know about files or filesystems outside the jail. Since no program or process in the jail knows about anything outside the jail, and cannot read or access anything outside the jail, the user is locked in. Not only can the client not break out of the jail, if the jail is hacked the intruder can't break out of the jail. This helps secure your system while meeting client needs.”[201] • “Each jail has its own /etc tree. While not everything in there is functional, it's simpler to ignore the extras than trim them out. You need to grab a copy of the /etc tree from the same source code you used to build your jail, and install it properly in the jail's directoryFinally, every FreeBSD system requires its own process filesystem, or procfs. If you're not using jails, you really don't need to worry about procfs; it appears automatically when you boot the system, cannot be tuned, and programs fairly transparently access it when

[198] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 261 (2002)
[199] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 261-62 (2002)
[200] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 176 (2002)
[201] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 176 (2002)

’058 Patent Claim 1	Disclosure
	needed. It's a necessary bit of infrastructure, however. I create a script /usr/local/etc/rc.d/jail.sh and add all the procfs mount lines to this script.”[202] • “You'll see a shell prompt, at which point you're in single−user mode in your jail and your jail is up and running. . . . Processes in the jail cannot see the rest of the system. Our host server is running a jail, among many other things. Here's a top snapshot from within a jail running in single−user mode. You can see that the shell process is running, and the top process, but nothing else. You cannot see the processes from the main system.”[203] • “From this point on, your jail will resemble a default FreeBSD install in which you can configure nameservice, add packages and users, and so on. Once you exit this shell, though, the virtual machine will stop running and your jail will shut down. . . . At this point, your jail is running. You can ssh in and configure it exactly as you would any other system.”[204] To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’058 Patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.
1[e][2] where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and	FreeBSD, as evidenced by the example citations below, discloses where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, as this claim limitation appears to be interpreted by VirtaMove. *See*, *e.g.*, the disclosures set forth at claim elements 1[c], 1[d][1], and 1[e][1]. This limitation is satisfied by the disclosures set forth at claim elements 1[c], 1[d][1], and 1[e][1]. To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’058 Patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.

[202] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 178-79 (2002)
[203] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 180 (2002)
[204] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 182 (2002)

’058 Patent Claim 1	Disclosure
1[f][1] iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and	FreeBSD, as evidenced by the example citations below, discloses wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, as this claim limitation appears to be interpreted by VirtaMove. *See, e.g.*, the disclosures set forth at claim elements 1[d][1]. *See also*, *e.g.*: • “FreeBSD has many noteworthy features. Some of these are: . . . *Preemptive multitasking* with dynamic priority adjustment to ensure smooth and fair sharing of the computer between applications and users, even under the heaviest of loads. . . . *Multiuser facilities* which allow many people to use a FreeBSD system simultaneously for a variety of things. This means, for example, that system peripherals such as printers and tape drives are properly shared between all users on the system or the network and that individual resource limits can be placed on users or groups of users, protecting critical system resources from over-use. . . . *Memory protection* ensures that applications (or users) cannot interfere with each other. One application crashing will not affect others in any way. . . .Thousands of *ready-to-run* applications are available from the FreeBSD ports and packages collection. . . . Thousands of additional and *easy-to-port* applications available on the Internet. FreeBSD is source code compatible with most popular commercial Unix systems and thus most applications require few, if any, changes to compile. . . . Demand paged *virtual memory* and ‘merged VM/buffer cache’ design efficiently satisfies applications with large appetites for memory while still maintaining interactive response to other users.”[205] • “Since our release of FreeBSD 2.0 in late 94, the performance, feature set, and stability of FreeBSD has improved dramatically. The largest change is a revamped virtual memory system with a merged VM/file buffer cache that not only increases performance, but also reduces FreeBSD's memory footprint, making a 5MB configuration a more acceptable minimum.”[206] • “In addition to the base distributions, FreeBSD offers a ported software collection with thousands of commonly sought-after programs. By mid-January 2000, there were nearly 3000 ports! The list of ports ranges from http (WWW) servers, to games, languages, editors, and almost everything in between. The entire ports collection requires approximately 50MB of storage, all ports being expressed as ‘deltas’ to their original sources. This makes it much easier for us to update ports, and greatly reduces the disk space demands made by the older 1.0 ports collection. To compile a port, you simply change to the directory of the program you wish to install, type make install, and let the system do the rest. The full original distribution for each port you build is retrieved dynamically off the CDROM or a local FTP site, so you need only enough disk space to build the ports you want. Almost every port is also provided as a pre-compiled ‘package’, which can be installed with a simple command (pkg_add) by those who do not wish to compile their own ports from source.”[207]

[205] https://web.archive.org/web/20000409013938/http://www6.freebsd.org/handbook/nutshell.html (emphasis in original)
[206] https://web.archive.org/web/20000815230603/http://www.freebsd.org/handbook/history.html
[207] https://web.archive.org/web/20000815230603/http://www.freebsd.org/handbook/history.html

’058 Patent Claim 1	Disclosure
	• “FreeBSD, having its history rooted in BSD UNIX, has its fundamentals based on several key UNIX concepts. The first, and most pronounced, is that FreeBSD is a multi-user operating system. The system can handle several users all working simultaneously on completely unrelated tasks. The system is responsible for properly sharing and managing requests for hardware devices, peripherals, memory, and CPU time evenly to each user.”[208] • “Because the system is capable of supporting multiple users, everything the system manages has a set of permissions governing who can read, write, and execute the resource. These permissions are stored as an octet broken into three pieces, one for the owner of the file, one for the group that the file belongs to, and one for everyone else. . . . This is all well and good, but how does the system control permissions on devices? FreeBSD actually treats most hardware devices as a file that programs can open, read, and write data to just like any other file. These special device files are stored on the /dev directory. Directories are also treated as files. They have read, write, and execute permissions. The executable bit for a directory has a slightly different meaning than that of files. When a directory is marked executable, it means it can be searched into, for example, a directory listing can be done in that directory.”[209] • “Since FreeBSD uses its file systems to determine many fundamental system operations, the hierarchy of the file system is extremely important. Due to the fact that the hier(7) man page provides a complete description of the directory structure, it will not be duplicated here. Please read hier(7) for more information. Of significant importance is the root of all directories, the / directory. This directory is the first directory mounted at boot time and it contains the base system necessary at boot time. The root directory also contains mount points for every other file system that you want to mount. A mount point is a directory where additional file systems can be grafted onto the root file system. Standard mount points include /usr, /var, /mnt, and /cdrom. These directories are usually referenced to entries in the file /etc/fstab. /etc/fstab is a table of various file systems and mount points for reference by the system. Most of the file systems in /etc/fstab are mounted automatically at boot time from the script rc(8) unless they contain the noauto option. Consult the fstab(5) manual page for more information on the format of the /etc/fstab file and the options it contains.”[210] • “The FreeBSD Ports collection allows you to compile and install a very wide range of applications with a minimum amount of effort. In general, it is a group of skeletons which contain a minimal set of items needed to make an application compile and install cleanly on FreeBSD. Even with all the hype about open standards, getting a program to compile on various UNIX platforms can be a tricky task. Occasionally, you might be lucky enough to find that the program you want compiles cleanly on your system, install everything into all the right directories, and run flawlessly ‘out-of-the-box’, but this behavior is somewhat rare. Most of the time, you find yourself needing to make modifications in order to get the program to work.

[208] https://web.archive.org/web/20000622040046/http://www.freebsd.org/handbook/permissions.html
[209] https://web.archive.org/web/20000622040046/http://www.freebsd.org/handbook/permissions.html
[210] https://web.archive.org/web/20000621185132/http://www.freebsd.org/handbook/dirstructure.html

’058 Patent Claim 1	Disclosure

	This is where the FreeBSD Ports collection comes to the rescue. The general idea behind the Ports collection is to eliminate all of the messy steps involved with making things work properly so that the installation is simple and very painless. With the Ports collection, all of the hard work has already been done for you, and you are able to install any of the Ports collection ports by simply typing make install.”[211]
	• “The first thing that should be explained when it comes to the Ports collection is what is actually meant by a ‘skeleton’. In a nutshell, a port skeleton is a minimal set of files that are needed for a program to compile and install cleanly on FreeBSD. Each port skeleton includes:
	  ○ A Makefile. The Makefile contains various statements that specify how the application should be compiled and where it should be installed on your system
	  ○ A files directory. The files directory contains a file named md5. This file is named after the MD5 algorithm used to determine ports checksums. A checksum is a number generated by adding up all the data in the file you want to check. If any characters change, the checksum will differ from the original and an error message will be displayed so you are able to investigate the changes.
	  ○ The files directory can also contain other files that are required by the port but do not belong elsewhere in the directory structure.
	  ○ A patches directory. This directory contains patches to make the program compile and install on your FreeBSD system. Patches are basically small files that specify changes to particular files. They are in plain text format, and basically say ‘Remove line 10’ or ‘Change line 26 to this ...’. Patches are also known as ‘diffs’ because they are generated by the diff program.
	  ○ A pkg directory. This directory normally contains three files. Occasionally, there will be more than three, but it depends on the port. Most only require three. The files are:
	  ▪ COMMENT. This is a one-line description of the program.
	  ▪ DESCR. This is a more detailed, often multiple-line, description of the program.
	  ▪ PLIST. This is a list of all the files that will be installed by the port. It also tells the ports system what files to remove upon deinstallation.”[212]
	• “FreeBSD uses a three-stage bootstrap by default, which basically entails three programs which call each other in order (two boot blocks, and the loader). Each of these three build on the previous program's understanding and provide increasing amounts of sophistication. The kernel is then started, which will then probe for devices and initialize them for use. Once the kernel boot process is finished, the kernel passes control to the user process init(8), which then makes sure the disks are in a usable state. init(8) then starts the user-level resource configuration which then mounts filesystems, sets up network cards to act on the network, and generally starts all the processes that usually are run on a FreeBSD system at startup.”[213]
	• “The easy-to-use command set comprises of:

[211] https://web.archive.org/web/20000815234714/http://www.freebsd.org/handbook/ports.html
[212] https://web.archive.org/web/20000815235607/http://www.freebsd.org/handbook/ports-using.html
[213] https://web.archive.org/web/20000815235626/http://www.freebsd.org/handbook/boot.html

’058 Patent Claim 1	Disclosure
	autoboot seconds Proceeds to boot the kernel if not interrupted within the time span given, in seconds. It displays a countdown, and the default timespan is 10 seconds. boot [-options] [kernelname] Immediately proceeds to boot the kernel, with the given options, if any, and with the kernel name given, if it is. boot-conf Goes through the same automatic configuration of modules based on variables as what happens at boot. This only makes sense if you use unload first, and change some variables, most commonly kernel. help [topic] Shows help messages read from /boot/loader.help. If the topic given is index, then the list of available topics is given. include filename ... Processes the file with the given filename. The file is read in, and interpreted line by line. An error immediately stops the include command. load [-t type] filename Loads the kernel, kernel module, or file of the type given, with the filename given. Any arguments after filename are passed to the file. ls [-l] [path] Displays a listing of files in the given path, or the root directory, if the path is not specified. If -l is specified, file sizes will be shown too. lsdev [-v] Lists all of the devices from which it may be possible to load modules. If -v is specified, more details are printed. lsmod [-v] Displays loaded modules. If -v is specified, more details are shown. more filename Display the files specified, with a pause at each LINES displayed. reboot Immediately reboots the system. set variable, set variable=value Set loader's environment variables. unload Removes all loaded modules.”[214]

[214] https://web.archive.org/web/20000815235638/http://www.freebsd.org/handbook/boot-loader.html

’058 Patent Claim 1	Disclosure
	• “Here are some practical examples of loader usage. ○ To simply boot your usual kernel, but in single-user mode: boot -s ○ To unload your usual kernel and modules, and then load just your old (or another) kernel: unload load kernel.old You can use kernel.GENERIC to refer to the generic kernel that comes on the install disk, or kernel.old to refer to your previously installed kernel (when you've upgraded or configured your own kernel, for example). Note: Use the following to load your usual modules with another kernel: unload set kernel=‘kernel.old’ boot-conf ○ To load a kernel configuration script (an automated script which does the things you'd normally do in the kernel boot-time configurator): load -t userconfig_script /boot/kernel.conf”[215] • “5.4. Kernel Interaction During Boot . . . Once the kernel is loaded by either loader (as usual) or boot2 (bypassing the loader), it examines its boot flags, if any, and adjusts its behavior as necessary.”[216] • “5.4.1. Kernel Boot Flags Here are the more common boot flags: -a during kernel initialization, ask for the device to mount as [] the root file system. -C boot from CDROM.

[215] https://web.archive.org/web/20000815235638/http://www.freebsd.org/handbook/boot-loader.html
[216] https://web.archive.org/web/20000815235643/http://www.freebsd.org/handbook/boot-kernel.html

’058 Patent Claim 1	Disclosure
	-c run UserConfig, the boot-time kernel configurator -s boot into single-user mode -v be more verbose during kernel startup”[217] • “5.5.2. Single-User Mode . . . This mode can be reached through the automatic reboot sequence, or by the user booting with the -s or setting the boot_single variable in loader. It can also be reached by calling shutdown without the reboot (-r) or halt (-h) options, from multi-user mode.If the system console console is set to insecure in /etc/ttys, then the system prompts for the root password before initiating single-user mode. . . . 5.5.3. Multi-User Mode . . . If init finds your filesystems to be in order, or once the user has finished in single-user mode, the system enters multi-user mode, in which it starts the resource configuration of the system. . . . 5.5.3.1. Resource Configuration (rc) . . . The resource configuration system reads in configuration defaults from /etc/defaults/rc.conf, and system-specific details from /etc/rc.conf, and then proceeds to mount the system filesystems mentioned in /etc/fstab, start up networking services, starts up miscellaneous system daemons, and finally runs the startup scripts of locally installed packages. rc(8) is a good reference to the resource configuration system, as is examining the scripts themselves.”[218] • “All access to the system is achieved via accounts, and all processes are run by users, so user and account management are of integral importance on FreeBSD systems. There are three main types of accounts; the Superuser, system users, and user accounts. The Superuser account, usually called root, is used to manage the system with no limitations on privileges. System users run services. Finally, user accounts are used by real people, who log on, read mail, and so forth.”[219] • “The superuser account, usually called root, comes preconfigured, and facilitates system administration, and should not be used for day-to-date tasks like sending and receiving mail, general exploration of the system, or programming. This is because the superuser, unlike normal user accounts, can operate without limits, and misuse of the superuser account may result in spectacular disasters. User accounts are unable to destroy the system by mistake, so it is generally best to use normal user accounts whenever possible, unless you especially need the extra privilege.”[220] • “System users are those used to run services such as DNS, mail, web servers, and so forth. The reason for this is security, as if all services ran as the superuser, they could act without restriction.”[221]

[217] https://web.archive.org/web/20000815235643/http://www.freebsd.org/handbook/boot-kernel.html
[218] https://web.archive.org/web/20000815235647/http://www.freebsd.org/handbook/boot-init.html
[219] https://web.archive.org/web/20000815235658/http://www.freebsd.org/handbook/users.html
[220] https://web.archive.org/web/20000815235703/http://www.freebsd.org/handbook/users-superuser.html
[221] https://web.archive.org/web/20000815235707/http://www.freebsd.org/handbook/users-system.html

’058 Patent Claim 1	Disclosure
	• “User accounts are the primary means of access for real people to the system, and these accounts insulate the user and the environment, preventing the users from damaging the system or other users, and allowing users to customize their environment without affecting others. Every person accessing your system should have their own unique user account. This allows you to find out who is doing what, and prevent people from clobbering each others’ settings, and reading mail meant for the other, and so forth. Each user can set up their own environment to accommodate their use of the system, by using alternate shells, editors, key bindings, and language.”[222] • “Almost every device in the kernel has a corresponding ‘node’ entry in the /dev directory. These nodes look like regular files, but are actually special entries into the kernel which programs use to access the device. The shell script /dev/MAKEDEV, which is executed when you first install the operating system, creates nearly all of the device nodes supported. However, it does not create all of them, so when you add support for a new device, it pays to make sure that the appropriate entries are in this directory, and if not, add them.”[223] • “The filesystem is best visualized as a tree, rooted, as it were, at /. /dev, /usr, and the other directories in the root directory are branches, which may have their own branches, such as /usr/local, and so on.”[224] • “During the boot process, filesystems listed in /etc/fstab are automatically mounted (unless they are listed with noauto). The /etc/fstab file contains a list of lines of the following format: device /mount-point fstype options dumpfreq passno device is a device name (which should exist), as explained in the Disk naming conventions above. mount-point is a directory (which should exist), on which to mount the filesystem. fstype is the filesystem type to pass to mount(8). The default FreeBSD filesystem is ufs. options is either rw for read-write filesystems, or ro for read-only filesystems, followed by any other options that may be needed. A common option is noauto for filesystems not normally mounted during the boot sequence. Other options in the mount(8) manual page.

[222] https://web.archive.org/web/20000815235712/http://www.freebsd.org/handbook/users-user.html
[223] https://web.archive.org/web/20000815235745/http://www.freebsd.org/handbook/kernelconfig-nodes.html
[224] https://web.archive.org/web/20000815235930/http://www.freebsd.org/handbook/disks-mounting.html

’058 Patent Claim 1	Disclosure
	dumpfreq is the number of days the filesystem should be dumped, and passno is the pass number during which the filesystem is mounted during the boot sequence.”[225] • “The mount(8) command is what is ultimately used to mount filesystems. . . . The umount command takes, as a parameter, one of a mountpoint, a device name, or the -a or -A option. . . . -a and -A are used to unmount all mounted filesystems, possibly modified by the filesystem types listed after -t. -A, however, doesn't attempt to unmount the root filesystem.”[226] • “jail -- imprison process and its descendants”[227] • “The jail command imprisons a process and all future decendants.”[228] • “A number of daemons ship with the base system that may have problems when run from outside of a jail in a jail-centric environment. This includes syslogd(8), sendmail(8), named(8), and portmap(8). While sendmail and named can be configured to listen only on a specific IP using their configuration files, in most cases it is easier to simply run the daemons in jails only, and not in the host environment. Syslogd cannot be configured to bind only a single IP, but can be configured to not bind a network port, using the ‘-ss’ argument. Attempting to serve NFS from the host environment may also cause confusion, and cannot be easily reconfigured to use only specific IPs, as some NFS services are hosted directly from the kernel. Any third party network software running in the host environment should also be checked and configured so that it does not bind all IP addresses, which would result in those services also appearing to be offered by the jail environments. Once these daemons have been disabled or fixed in the host environment, it is best to reboot so that all daemons are in a known state, to reduce the potential for confusion later (such as finding that when you sendmail to a jail, and its sendmail is down, the mail is delivered to the host, etc.)”[229] • “Once a process has been put in a prison, it and its decendants cannot escape the prison. It is not possible to add a process to a preexisting prison. Inside the prison, the concept of ‘superuser’ is very diluted. In general, it can be assumed that nothing can be mangled from inside a prison which does not exist entirely inside that prison. For instance[,] the directory tree below ‘path’ can be manipulated all the ways a root can normally do it, including ‘rm-rf /*’ but new device special nodes cannot be created because they reference shared resources (the device drivers in the kernel).”[230]

[225] https://web.archive.org/web/20000815235930/http://www.freebsd.org/handbook/disks-mounting.html
[226] https://web.archive.org/web/20000815235930/http://www.freebsd.org/handbook/disks-mounting.html
[227] https://web.archive.org/web/20210506161927/https://www.freebsd.org/cgi/man.cgi?query=jail&apropos=0&sektion=0&manpath=FreeBSD+4.0-RELEASE&arch=default&format=html
[228] https://web.archive.org/web/20210506161927/https://www.freebsd.org/cgi/man.cgi?query=jail&apropos=0&sektion=0&manpath=FreeBSD+4.0-RELEASE&arch=default&format=html
[229] https://web.archive.org/web/20210506161927/https://www.freebsd.org/cgi/man.cgi?query=jail&apropos=0&sektion=0&manpath=FreeBSD+4.0-RELEASE&arch=default&format=html
[230] https://web.archive.org/web/20220508073711/https://www.freebsd.org/cgi/man.cgi?query=jail&sektion=2&apropos=0&manpath=FreeBSD+4.0-RELEASE

Chart B-10 **Invalidity Contentions: FreeBSD**

'058 Patent Claim 1	Disclosure
	• "The FreeBSD 'Jail' facility provides the ability to partition the operating system environment, while maintaining the simplicity of the UNIX 'root' model. In Jail, users with privilege find that the scope of their requests is limited to the jail, allowing system administrators to delegate management capabilities for each virtual machine environment. Creating virtual machines in this manner has many potential uses; the most popular thus far has been for providing virtual machine services in Internet Service Provider environments."[231] • "To this end, we describe the new FreeBSD 'Jail' facility, which provides a strong partitioning solution, leveraging existing mechanisms, such as chroot(2), to what effectively amounts to a virtual machine environment. Processes in a jail are provided full access to the files that they may manipulate, processes they may influence, and network services they can make use of, and neither access nor visibility of files, processes or network services outside their partition."[232] • "Jail neatly side-steps the majority of these problems through partitioning. Rather than introduce additional fine-grained access control mechanism, we partition a FreeBSD environment (processes, file system, network resources) into a management environment, and optionally subset Jail environments. . . . Consequently[,] the administrator of a FreeBSD machine can partition the machine into separate jails, and provide access to the super-user account in each of these without losing control of the over-all environment."[233] • "A process in a partition is referred to as 'in jail'. When a FreeBSD system is booted up after a fresh install, no processes will be in jail. When a process is placed in a jail, it, and any descendents of the process created after the jail creation, will be in that jail. A process may be in only one jail, and after creation, it [cannot] leave the jail. Jails are created when a privileged process calls the jail(2) syscall, with a description of the jail as an argument to the call. Each call to jail(2) creates a new jail; the only way for a new process to enter the jail is by inheriting access to the jail from another process already in that jail. Processes may never leave the jail they created, or were created in." [234] • "Jail takes advantage of the existing chroot(2) behaviour to limit access to the file system name-space for jailed processes. When a jail is created, it is bound to a particular file system root. Processes are unable to manipulate files that they cannot address, and as such the integrity and confidentiality of files outside of the jail file system root are protected. Traditional mechanisms for breaking out of chroot(2) have been blocked. In the expected and documented configuration, each jail is provided with its exclusive file system root, and standard FreeBSD directory layout, but this is not mandated by the implementation."[235]

[231] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[232] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[233] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[234] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[235] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf

Chart B-10 **Invalidity Contentions: FreeBSD**

’058 Patent Claim 1	Disclosure
	• “While the jail(2) call could be used in a number of ways, the expected configuration creates a complete FreeBSD installation for each jail. This includes copies of all relevant system binaries, data files, and its own /etc directory. Such a configuration maximises the independence of various jails, and reduces the chances of interference between jails being possible, especially when it is desirable to provide root access within a jail to a less trusted user.”[236] • “The jail environments must also be custom-configured. This consists of building and installing a miniature version of the FreeBSD file system tree off of a subdirectory in the host environment, usually /usr/jail, or /data/jail, with a subdirectory per jail. Appropriate instructions for generating this tree are included in the jail(8) man page, but generally this process may be automated using the FreeBSD build environment. One notable difference from the default FreeBSD install is that only a limited set of device nodes should be created. MAKEDEV(8) has been modified to accept a ‘jail’ argument that creates the correct set of nodes. To improve storage efficiency, a fair number of the binaries in the system tree may be deleted, as they are not relevant in a jail environment. This includes the kernel, boot loader, and related files, as well as hardware and network configuration tools.” [237] • “As it stands, the jail code provides a strict subset of system resources to the jail environment, based on access to processes, files, network resources, and privileged services.”[238] • “The jail facility provides FreeBSD with a conceptually simple security partitioning mechanism, allowing the delegation of administrative rights within virtual machine partitions. The implementation relies on restricting access within the jail environment to a well-defined subset of the overall host environment. This includes limiting interaction between processes, and to files, network resources, and privileged operations. Administrative overhead is reduced through avoiding fine-grained access control mechanisms, and maintaining a consistent administrative interface across partitions and the host environment.”[239] • “Your system should have fast access to its root filesystem (/), which contains the kernel and just enough utilities and programs to boot the computer into its most basic running status, single−user mode.”[240] • “You should do everything possible while signed on as a regular user, and only use the root account when you must change the system. That will happen frequently at first, but will grow less common as time passes. Before you can sign on as a regular user, though, you need to set one up for your use.” [241] • “The ‘Home directory’ is where the user's files are kept. The default is generally fine.” [242]

[236] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[237] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[238] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[239] https://papers.freebsd.org/2000/phk-jails.files/sane2000-jail.pdf
[240] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* (2002)
[241] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* (2002)
[242] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 214 (2002)

’058 Patent Claim 1	Disclosure
	• “Whenever your system boots to the point where it can execute userland commands, it runs the shell script /etc/rc. This script mounts all filesystems, brings up the network interfaces, configures device nodes, sets up shared libraries, and does all the other tasks required to set up a system.” • “FreeBSD can run a wide variety of software packages, most of which are available as source code so they can be built as native FreeBSD software. And, thanks to some clever design, FreeBSD can also run software from many foreign operating systems. We'll look at how to do this, focusing on the popular Linux compatibility package that allows FreeBSD to run unmodified Linux software.”[243] • “Managing Shared Libraries . . . The basic idea behind a shared library is quite straightforward: It's a chunk of compiled code that provides services and functions to other chunks of compiled code. Shared libraries provide popular functions for all programs to use, and they are designed to be reused by as many different programs as possible. For example, many programs must hash (or one−way encrypt) data as part of their function. But if every program had to include hashing code, each would be larger, harder to write, and more unpleasant to maintain. What's more, programs would have interoperability problems if they implemented hashes differently. By using a shared library (in this example, libcrypt), a program that needs hashing has access to the functions while eliminating problems of maintenance and interoperability. Similarly, other shared libraries provide common functions to support other software. This reduces the average size of programs, freeing up a reasonably large amount of system memory. FreeBSD builds a cache of available shared libraries at boottime. Programs don't have to scan the whole disk looking for shared libraries; they just ask the cache for the functions they want. In fact, the ability to manage the library cache is one thing that separates a newbie from a professional.”[244] • “When you execute an ELF binary that needs shared libraries, the system calls rtld(1), the ‘run−time linker.’ Rtld examines binaries as they're loaded, determines which shared libraries they need, and loads those libraries. . . . Rather than searching the entire system for anything that looks like a shared library everytime anything is executed, rtld pulls the shared libraries from a library cache. The cache lives on your system in two separate files: /var/run/ld.so.hints (aout) and /var/run/ld−elf.so.hints (ELF). A misconfigured cache is the most likely cause of shared library problems.”[245] • “To see the list of libraries you already have, run ldconfig with the −r flag:

[243] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 250 (2002)
[244] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 252 (2002)
[245] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 253 (2002)

’058 Patent Claim 1	Disclosure

	(Add)
	`fig -r`
	`/var/run/ld-elf.so.hints:`
	`search directories:`
	`/usr/lib:/usr/lib/compat:/usr/X11R6/lib:/usr/local/lib:/usr/local/lib/mysql:/usr/local/pilot/li`
	`0:-lcom_err.2 => /usr/lib/libcom_err.so.2`
	`1:-lscrypt.2 => /usr/lib/libscrypt.so.2`
	`2:-lcrypt.2 => /usr/lib/libcrypt.so.2`
	`...`
	
	ldconfig −r examines the shared library cache and lists every shared library it finds. On my system, this list runs to 229 shared libraries. ”[246]
	• “While the −m option works very well if you're the systems administrator, it won't work if you're just a lowly user without root access. Also, if you have your personal set of shared libraries, your sysadmin won't want to make them globally available, and root must own the shared library directory so that regular users can't just dump things in there willy−nilly. Sysadmins probably won't even want to take the slightest chance of system programs linking against your personal libraries. Here's where the LD_LIBRARY_PATH environment variable appears. Rather than create a cache, LD_LIBRARY_PATH tells the system to check the directories it lists for new shared libraries. . . . You can specify any number of directories in LD_LIBRARY_PATH, separated with colons. For example, I might want to put the directories /home/mwlucas/lib and /compat/linux/usr/lib/local into my LD_LIBRARY_PATH to complete a software install.” [247]
	• “Lastly, there's the question of what libraries a program expects to have available. You can get this information with ldd(1). This output tells us the names of the shared libraries Emacs requires, and the locations of the files that contain those libraries. You can check this list of required libraries against the output of ldconfig −r to confirm that your program has what it needs. Or you can use this as a shopping list and then go out and get the needed libraries.”[248]
	• “In addition to recompiling and emulating, the final option for running foreign programs is the one FreeBSD is best known for: *ABI (application binary interface) implementation*. The ABI is the part of the kernel that provides services to programs, including everything from sound−card access to reading files to printing on the screen to starting other programs—all the things a program needs to run. As far as programs are concerned, the ABI is the operating system. By completely implementing the ABI from a different operating system on your operating system, you can run non−native programs as if they were on the native platform. Or you can provide multiple ABIs and control which ABI a program uses. This is what FreeBSD does.”[249]

[246] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 253 (2002)
[247] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 254-55 (2002)
[248] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 255 (2002)
[249] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 257 (2002)

’058 Patent Claim 1	Disclosure
	• “Foreign Software Libraries While the kernel portion of the ABI solves one major issue, the other portions of the system are another problem because every operating system has its own requirements in addition to the kernel. The biggest issue is shared libraries. If the kernel starts a program, and the program can't find its shared libraries, it won't work correctly. No matter which ABI you use, you must have a copy of the shared libraries for that platform. SVR4 and SCO For example, to use the SVR4 and SCO ABIs, you need access to the appropriate system. While a Sun Solaris 2.6 CD will suffice for the SVR4 module, you need to grab the shared libraries from an actual SCO UNIX machine to use the SCO ABI, which means you need a SCO or Solaris license. This isn't an insurmountable problem, of course, but it does make using this module slightly more difficult—and definitely more expensive. OSF/1 A minimal set of OSF/1 shared libraries are available under /usr/ports/emulators/osf1_base. These libraries have a restrictive license and can only be used in fairly narrow circumstances, but you can get a more complete set of shared libraries from an actual OSF/1 system, if you wish. If you have an actual OSF/1 license, you can pretty much do whatever you like with the libraries Linux The shared libraries for the Linux mode are the most freely available of any mode. Since the barrier to entry is so low, we'll discuss Linux compatibility in some detail. Once you have a thorough understanding of how it works, you can apply this knowledge to any other ABI compatibility you need to implement. If you have a spare Alpha lying around (other than the Multia model known for Random Hea Death), feel free to ship it to me in care of No Starch Press; I'll be delighted to include a discussion of OSF/1 mode in the next edition of Absolute BSD.”[250] • “Alternatively, suppose a Linux binary wants to call sh(1). The Linux ABI will first check under /usr/compat/linux and find bin/sh. When it finds sh there, it will execute that program instead of the FreeBSD native /bin/sh.”[251]

[250] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 259 (2002)
[251] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 261 (2002)

’058 Patent Claim 1	Disclosure
	• “As I mentioned earlier, the Linux install in linux_base is rather minimal, and some Linux programs expect a broader range of shared libraries to be available. FreeBSD tries to keep ports as small as possible, but compromises by making these additional Linux libraries available as additional ports. The ports. collection includes several ports that augment linux_base, most of which are shared libraries. These increase the range of programs that FreeBSD's Linux mode can support. Installing these ports will round out 99 percent of the functionality you might want to provide to a Linux program. You may need a shared library or program that is not available in linux_base or a port. If so, the simplest thing to do is to find a Linux system of the appropriate version, copy the files, and install them in the appropriate locations under /usr/compat/linux. If you've created a new directory to contain shared libraries, you'll need to tweak /usr/compat/linux/etc/ld.so.conf.”[252] To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’058 Patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.
1[f][2] wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.	FreeBSD, as evidenced by the example citations below, discloses wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously, as this claim limitation appears to be interpreted by VirtaMove. *See*, *e.g.*, the disclosures set forth at claim elements 1[c], 1[d][1], 1[e][1], and 1[f][1]. This limitation is satisfied by the disclosures set forth at claim elements 1[c], 1[d][1], 1[e][1], and 1[f][1]. To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant’s Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the ’058 Patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.

[252] Lucas, M., *Absolute BSD: the Ultimate Guide to Free BSD* 261-62 (2002)

Chart B-10 **Invalidity Contentions: FreeBSD**

Claim 2

'058 Patent Claim 2	**Disclosure**
2. A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.	FreeBSD, as evidenced by the example citations below, discloses a computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system, as this claim limitation appears to be interpreted by VirtaMove. *See, e.g.,* the disclosures set forth at claim elements 1[c] and 1[f][1]. This element is satisfied by the disclosure set forth at claim elements 1[c] and 1[f][1]. To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant's Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the '058 Patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.

Chart B-10 **Invalidity Contentions: FreeBSD**

Claim 3

'058 Patent Claim 3	**Disclosure**
3. A computing system according to claim 1 wherein OSC SES corresponding to and capable of performing the same function as SLCSES remain in the operating system kernel.	FreeBSD, as evidenced by the example citations below, discloses a computing system according to claim 1 wherein OSCSES corresponding to and capable of performing the same function as SLCSES remain in the operating system kernel, as this claim limitation appears to be interpreted by VirtaMove. *See, e.g.,* the disclosures set forth at claim element 1[c] and 1[d][1]. This element is satisfied by the disclosure set forth at claim element 1[c] and 1[d][1]. To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant's Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the '058 Patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.

Chart B-10 **Invalidity Contentions: FreeBSD**

Claim 4

'058 Patent Claim 4	**Disclosure**
4. A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.	FreeBSD, as evidenced by the example citations below, discloses a computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of Software applications having exclusive use thereof, use system calls to access services in the operating system kernel, as this claim limitation appears to be interpreted by VirtaMove. *See, e.g.,* the disclosures set forth at claim elements 1[b], 1[c], and 1[d][1]. This element is satisfied by the disclosures set forth at claim elements 1[b], 1[c], and 1[d][1]. To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant's Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the '058 Patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.

Chart B-10 **Invalidity Contentions: FreeBSD**

Claim 18

'058 Patent Claim 18	**Disclosure**
18. A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.	FreeBSD, as evidenced by the example citations below, discloses the computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs, as this claim limitation appears to be interpreted by VirtaMove. *See, e.g.,* the disclosures set forth at claim elements 1[b]-[d][1]. This element is satisfied by the disclosures set forth at claim elements 1[b]-[d][1]. To the extent FreeBSD does not expressly disclose this limitation, a person of ordinary skill in the art would have determined that this limitation is either inherent and/or obvious to one of ordinary skill in the art in view of the teachings of FreeBSD. Further, one of ordinary skill in the art would have been motivated to modify FreeBSD or combine it with any of the present prior art references found in Defendant's Invalidity Contentions and any supplements thereto and the relevant section of charts for other prior art for the '058 Patent in a manner that would result in the subject matter of this limitation given, at the very least, the person's understanding of the state of the art, the problems addressed and solved in the prior art, and the teachings of FreeBSD.